MASTER EN SCIENCES INFORMATIQUES

La réalisation d'un dossier médical portable : Websanté

Scieur, Benoît

Award date:
1995

Awarding institution:
Universite de Namur

FACULTES
UNIVERSITAIRES
N.D. DE LA PAIX

NAMUR

FUNDP
NAMUR

INSTITUT D'INFORMATIQUE

La réalisation d'un dossier médical portable : WebSanté

Benoît Scieur

Promoteur : Mr. Jean Ramaekers

Mémoire présenté en vue de l'obtention du titre de maître en informatique

Année académique 1995-1996

RUE GRANGAGNAGE, 21, B - 5000 NAMUR (BELGIUM)

LBS 685023

307371

Abstract

Smart card are more and more used for any type of applications. It gives some security features. Nevertheless, one problem remains : the lack of memory space. Our WebSanté application proposes a solution to this problem. The WebSanté card is a health portable file, that contains medical information about the card's carrier. For example, the pathologies the patient has suffered from, the elementary medical exams for his different pathologies, the services and the hospital where he went through. In the medical frame, other voluminous informations are needed : reports of examinations, radiographs, ... This cannot be loaded in a smart card. That's the reason why we convert them to HTML documents which we then index in the Card. The card becomes a secure tool for the navigation among the patient's files. Information can be modified and updated in the card thanks to a Web browser.

Key-words : Smart card, portable file, medical application, World Wide Web.

Résumé

Les cartes à microprocesseur sont de plus en plus utilisées pour tout type d'applications. Elles offrent un certain nombre de garanties au point de vue de la sécurité. Un problème majeur subsiste néanmoins : la faible place en mémoire. Notre application WebSanté propose une solution à ce problème. La carte WebSanté est un dossier portable médical qui à ce titre contient des informations médicales concernant le porteur de la carte (les pathologies dont souffre le patient, les actes élémentaires effectués dans le cadre de ses pathologies et les services et les hôpitaux par lesquels le patient est passé, ...). Dans le cadre médical, d'autres informations volumineuses sont intéressantes : les rapports des examens, les radiographies, ... Ces informations ne pouvant entrer dans la carte, nous les transformons en document HTML que nous pouvons indexer dans la carte. Celle-ci est alors un support sécurisé de navigation parmi les dossiers du patient. Toutes les modifications et toutes les mises à jour des informations contenues dans la carte sont effectuées par un browser Web.

Mots clés : carte à microprocesseur, dossier portable, application médicale, World Wide Web.

Remerciements

La réalisation d'un mémoire ne se fait jamais sans la collaboration d'un certain nombre d'individus. Je tiens donc à remercier l'ensemble des personnes qui de près ou de loin m'ont aidé et soutenu tout au long de ce travail.

Tout d'abord, je pense avoir bénéficié d'un cadre formidable pour mon stage : RD2P. Je tiens à souligner l'accueil et tous les petits coups de mains qui m'ont été fournis au jour le jour par chacun des membres de RD2P. Je tiens à remercier tout particulièrement Jean-Jacques Vandewalle et Pierre Paradinas pour leur suivi quotidien.

Un grand merci à mon promoteur Monsieur Jean Ramaekers pour ses éclairages et ses conseils lors de la rédaction. Merci une fois de plus à Jean-Jacques pour les nombreuses heures qu'il a passées à relire mon travail. Merci aussi à Pascal Goossens pour les ultimes conseils qu'il m'a prodigués.

Enfin, je remercie Jacques Scieur et Pierre Scieur qui ont joué le rôle de vérificateurs orthographique et grammairien.

Table des matières

Table des figures

Table des tableaux

Introduction générale

Dans ce travail, nous allons vous présenter notre application WebSanté. Il s'agit d'un **dossier portable sur carte à microprocesseur**. Les informations contenues dans la carte sont de type médical. Notre objectif est de modéliser le comportement du patient dans un milieu hospitalier. Chaque patient possède sa carte. Le personnel médical peut consulter et mettre à jour les informations contenues dans la carte du patient.

Le premier chapitre nous fait découvrir **le contexte du projet**. Dans celui-ci, nous introduisons d'une façon générale les principes et les objectifs de notre application. Ainsi, nous expliquons, dans un premier temps, le cadre dans lequel notre stage s'est déroulé. Ensuite, nous énonçons le cahier des charges qui a été réalisé. Nous terminons par une présentation des problématiques auxquelles nous devrons faire face.

Dans la perspective de la réalisation d'une application portant sur un dossier portable, notre premier centre d'intérêt est **la carte à microprocesseur**. Celle-ci est l'objet du second chapitre. D'une part, nous exposons les notions de base concernant les cartes à microprocesseur. D'autre part, nous présentons précisément la carte que nous utilisons pour notre application : **la carte CQL**[1]. Comme nous aurons l'occasion de le constater dans ce chapitre, la structure de la carte CQL est basée sur des tables et son langage d'interrogation est un sous-ensemble du langage SQL.

Le troisième chapitre fournit une introduction au **World Wide Web**. Notre application WebSanté utilise l'Internet comme une interface et comme système d'indexation. Les documents médicaux sont bien souvent trop volumineux pour être introduits dans une carte à microprocesseur. De là provient l'idée que la carte contiendrait uniquement des index vers ces documents volumineux. L'Internet fournit une interface conviviale. Et c'est donc celle-ci que nous utilisons pour insérer et mettre à jour des informations dans la carte.

[1] CQL est une marque déposée du groupe Gemplus.

Dans le quatrième chapitre, nous entrons dans **le développement du logiciel** proprement dit. Nous nous attardons tout d'abord sur la méthode que nous avons employée lors de la réalisation de cette application. La première étape consiste en la spécification du système d'information pour un patient en milieu hospitalier. Ensuite, nous présentons la conception physique. La dernière étape concerne l'implémentation et comprend notamment un modèle de code source et un scénario de consultation de la carte.

Le dernier chapitre nous permet d'évoquer deux problèmes qui nous sont apparus au cours de la réalisation du logiciel. Le premier concerne **la sécurité**. Vous pourrez constater par vous-même dans le second chapitre que les cartes à microprocesseur garantissent un bon niveau de sécurité. Mais qu'en est-il de l'Internet ? Une fois que nous allons chercher un document qui est indexé dans la carte, il circule sans protection sur le réseau. C'est pourquoi, nous présentons un protocole particulier qui sécurise les transactions sur l'Internet. Le deuxième problème que nous avons rencontré est celui de l'utilisation d'une plate-forme Orbix pour se connecter à la carte. Il n'est pas réaliste d'imaginer que chaque ordinateur des membres du personnel médical possède Orbix. Nous proposons une solution basée sur **les sockets Unix** afin de remplacer Orbix.

De cette façon, nous avons présenté toutes les facettes de notre application. Nous sommes conscients que certains éléments doivent encore être modifiés avant d'aboutir à un produit mûr. Toutefois, l'objectif essentiel de notre application est de montrer qu'un tel produit est réalisable.

Chapitre I :
Le contexte du projet

1. Le cadre de travail

Recherche et Développement Dossier Portable (RD2P) est un laboratoire de recherche dont la spécialité est la carte à microprocesseur. RD2P est la fusion de quatre entités : l'université de Lille 1 (scientifique), l'université de Lille 2 (médecine), le Centre National de Recherche Scientifique (CNRS) et la société Gemplus.

Chaque partenaire apporte sa spécificité aux recherches. Les professeurs et les maîtres de conférence de RD2P encadrent les étudiants provenant en majorité de l'université de Lille 1. Il s'agit d'étudiants qui y réalisent leur stage de fin d'étude ou leur thèse. L'université de Lille 2 intervient dans le fonctionnement de RD2P par l'intermédiaire du Centre de Recherche en Informatique Médicale (CERIM). Le CNRS, quant à lui, finance certains projets de RD2P. Gemplus est le partenaire industriel de RD2P; à ce titre, il apporte un soutien logistique, financier et humain.

Gemplus est une des plus importantes sociétés de production de cartes à microprocesseur dans le monde. Gemplus est une multinationale dont le siège social est situé à Gémenos (près de Marseille). Sa production mensuelle de cartes à microprocesseur est en augmentation constante. En 1992, elle produisait 5 millions de cartes par mois. En 1994, elle atteignait les 15 millions. Et fin 1995, sa production mensuelle était de 25 millions de cartes. L'objectif est d'atteindre rapidement les 600 millions de cartes par an. Gemplus est actuellement représenté par 25 succursales à travers le monde. Si l'on prend l'exemple des télécartes (les cartes de téléphone), Gemplus possède 40 % des parts de marché dans le monde et vend ses télécartes dans 50 pays. En 1995, 170 millions de télécartes Gemplus furent vendues à travers le monde.

Mon stage à RD2P s'insère dans la thèse de Jean-Jacques Vandewalle consacrée à l'intégration des cartes dans les systèmes d'information.

2. Le cahier des charges

2.1. Introduction

Depuis une dizaine d'années, les cartes à microprocesseur sont de plus en plus utilisées dans une grande diversité d'applications (bancaire, santé, ...). La limitation la plus souvent rencontrée, quand on travaille avec les cartes à microprocesseur, réside dans **la faible place**

mémoire disponible. Les cartes médicales souffrent tout particulièrement de cette insuffisance. En effet, les données à caractère médical sont très nombreuses et nécessitent **un espace de stockage volumineux**.

Notre objectif est de montrer qu' il est tout a fait réaliste de profiter d'autres ressources pour stocker les informations pertinentes dans le cas d'une application utilisant une carte à microprocesseur. Nous nous attacherons à démontrer qu'il est très intéressant de s'appuyer sur **le monde de l'Internet**. Et ce, tout d'abord, afin de profiter de l'indexation des ressources dans l'Internet. Ensuite, parce que l'Internet offre une interface et un accès pour le moins convivial. En effet, nous nous situons dans le domaine hospitalier et il est donc nécessaire que l'utilisation de notre logiciel soit la plus simple possible.

L'approche que nous allons suivre tout au long de ce travail nous permettra d'intégrer à travers la carte à microprocesseur une quantité importante de données. Nous utilisons pour ce faire une technique **d'indexation des ressources**. En effet, plutôt que de placer les données en tant que telles dans la carte, nous les référençons comme un document du monde de l'Internet. Cela représente un gain de place considérable dans la carte. Cela permet aussi d'y référencer des données qui auparavant n'étaient pas accessibles dans une carte. Nous entrons de plain-pied dans le domaine de l'hypermédia : des documents textuels, sonores et des images.

Dans le même ordre d'idée, il n'est pas irréaliste d'imaginer que l'on pourra dans un avenir proche insérer des données d'un type plus riche encore : une bande vidéo, une image en trois dimensions, ... D'ailleurs, ces médias commencent à se développer sur l'Internet. Ils représenteront un atout non négligeable pour les médecins qui pourront revivre des opérations ou des diagnostics d'autres collègues comme s'ils y étaient.

Avant de commencer toute analyse du problème, nous avons réalisé un cahier des charges. Celui-ci comporte, en premier lieu, une explication du milieu concerné par l'application. Ensuite, il comprend la lecture d'informations, l'écriture d'informations et des composants techniques.

2.2. Le milieu

L'application que nous avons développée modélise la gestion d'un patient dans un hôpital. Ce patient y est pris en charge dès son arrivée. Deux cas de figures sont possibles. Soit il entre à l'hôpital dans une situation normale, soit il est admis en urgence.

Nous avons pris en compte trois types de professions pour les personnes en contact avec le patient : les médecins, les infirmières et les hôtesses. Nous parlons ici des médecins au sens large ; aussi bien un médecin interne, qu'un urgentiste, qu'un dentiste, ... D'une façon générale, ce sont les personnes habilitées à effectuer des actes médicaux sur le patient. Les infirmières représentent le personnel soignant. Les hôtesses sont les personnes qui accueillent le patient lors de son arrivée (dans une situation normale). Elles ont une mission avant tout administrative.

2.3. Lecture d'informations

- Un professionnel de santé possède un ordinateur. Celui-ci contient un browser standard du type Netscape. Chaque professionnel dipose d'une page Web qui contient un lien vers le programme de gestion des cartes de ses patients.
- Le professionnel introduit la carte du patient dans le lecteur de carte. Il s'authentifie auprès de la carte. Un menu contextuel apparaît à l'écran. En effet, suivant que le professionnel est un médecin, une infirmière ou une hôtesse, il a des possibilités d'accès différentes sur la carte. Le dossier médical contient un certain nombre de données personnelles qu'il s'avère nécessaire de protéger des regards indiscrets.
- Le professionnel peut maintenant accéder aux informations disponibles pour le patient concerné. Chaque page de présentation contient des liens. Ces liens servent à accéder soit à une information contenue dans la carte soit à accéder à une information contenue dans un site distant.
- Les informations auxquelles le personnel médical accède peuvent aussi bien se présenter sous la forme de textes que d'images ou de sons.

2.4. Ecriture d'informations

- Avant toute modification, le professionnel de santé doit s'authentifier. De plus, suivant son statut, il pourra modifier certaines informations. Par exemple, une infirmière ne peut pas modifier le rapport d'un médecin. Par contre, le médecin pourra insérer tout nouvel acte qu'il aura posé sur le patient.
- L'information dans la carte doit être structurée. Cette structure doit permettre de retrouver les références disponibles pour un objet.

2.5. Les contraintes techniques

- Nous devons travailler sous Unix.
- Les cartes et les lecteurs de carte proviennent de la firme Gemplus.
- Pour accèder aux informations contenues dans la carte, nous utilisons un browser standard de type Netscape. Une condition importante est de ne modifier ni le browser ni le serveur HTTP. Il faut développer une solution accessible à tous sans contrainte particulière.

3. Explication des différentes problématiques

La *Figure 1* permet de se rendre compte que notre application touche un grand nombre de mondes différents. Dans cette section, nous introduisons brièvement les différents domaines qui sont concernés par notre application et qui seront développés dans les chapitres suivants.

Le principe de base est de **relier le professionnel de la santé à un serveur du monde Internet** (appelé serveur HTTP). Il possède un lecteur de carte et la carte du patient. La carte à microprocesseur est la composante centrale de notre application. Le chapitre II étudiera les cartes à microprocesseur.

Regardons un instant le comportement du système en cas d'accès à une donnée. La carte comprend outre les données standards d'une carte santé, des références vers des documents de l'Internet. Typiquement, les données standards d'une carte santé sont les suivantes : des informations administratives sur le patient, les actes médicaux entrepris sur celui-ci, les hôpitaux dans lesquels il a séjourné, ... Si le professionnel désire accéder à une donnée standard de la carte, c'est-à-dire une donnée entièrement située dans la carte, le cheminement est le suivant :

1. La demande est transférée du professionnel vers le serveur HTTP.
2. Un script gère cette demande.
3. Il accède aux données dans la carte.
4. Il les affiche au professionnel.

Au cas où le professionnel désire accéder à une donnée référencée dans la carte, le parcours diffère légèrement :

1. La demande est transférée du professionnel vers le serveur HTTP.
2. Le script accède à la donnée, située dans la carte, qui représente la référence d'un document de l'Internet.
3. Le serveur HTTP accède via le réseau Internet à ce document.
4. Ce document est affiché chez le client.

Nous avons choisi de travailler dans le monde de l'Internet non seulement car il nous permettait une indexation aisée des documents mais également pour sa relative facilité d'utilisation par des personnes peu habituées. Le chapitre trois nous introduira au monde de l'Internet : le Web, le langage HTML (HyperText Markup Language). Nous nous attarderons ensuite un instant sur les scripts, appelés des Common Gateway Interface (CGI), qui permettent de gérer les enchaînements des écrans et également dans notre cas les accès à la carte.

Ensuite, nous expliquerons, dans un quatrième chapitre, la communication entre la carte et les scripts. En effet, un lecteur de carte possède comme tout appareil d'entrée/sortie un numéro de port. Le problème est que la machine SERVEUR n'est pas au courant de ce numéro. Par ailleurs, le gestionnaire du lecteur de carte nécessite une connexion permanente avec la carte. Il n'est pas possible d'effectuer une demande à la carte, de couper la connexion et d'effectuer une autre demande par la suite. La connexion devrait être établie à nouveau avant tout transfert. Pour ces deux raisons, nous avons établi une communication client/serveur entre la carte et la machine SERVEUR. Cette connexion utilise la plate-forme Orbix, une implémentation de la norme Corba.

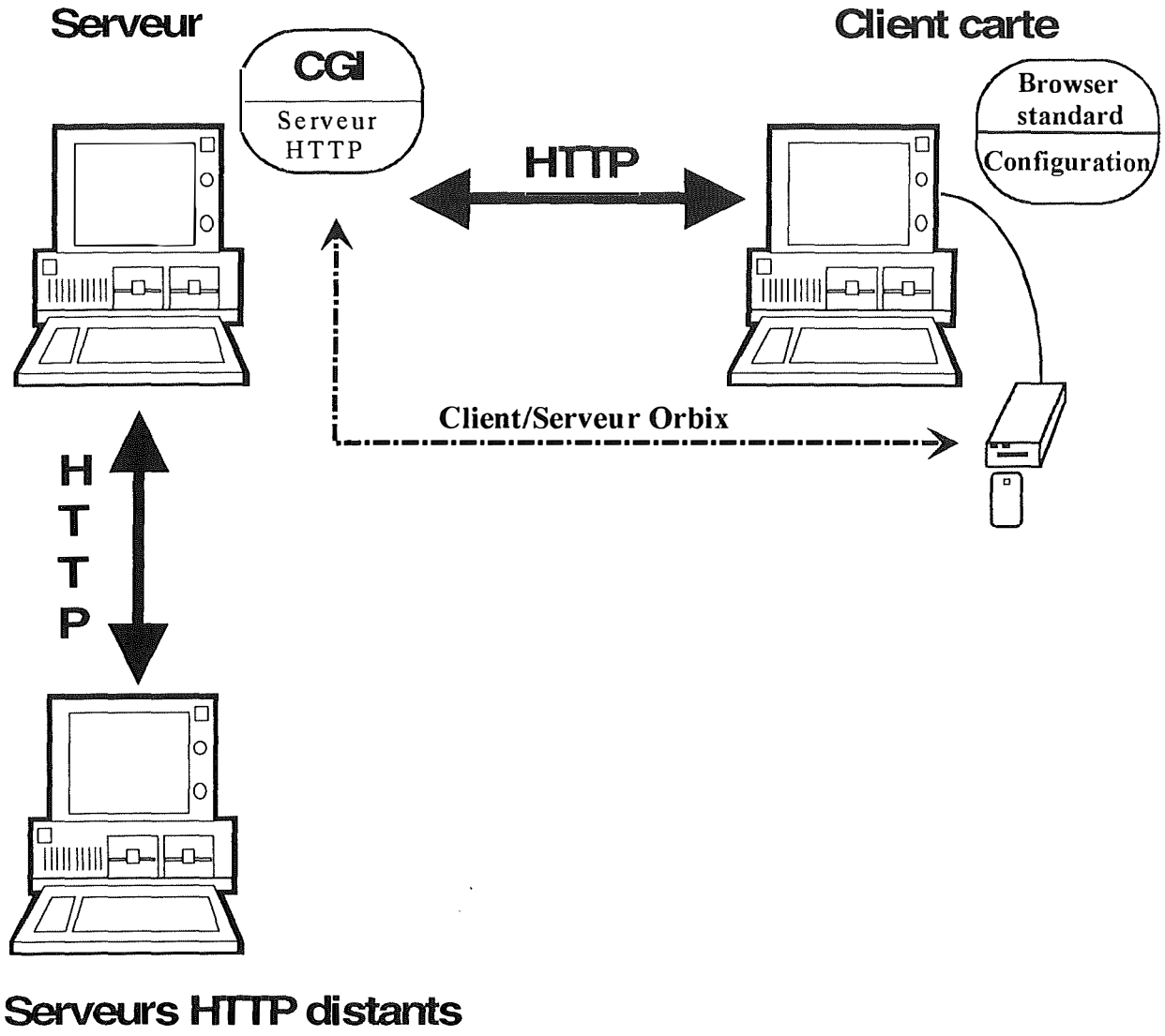


Figure 1: Le schéma global de l'architecture

Chapitre II :
La carte à microprocesseur

1. Introduction

Ce deuxième chapitre traite des cartes à microprocesseur. Dans une première section, nous abordons les notions de base de la carte à microprocesseur. Nous envisageons son historique, le cycle de vie d'une carte à microprocesseur, son architecture, sa sécurité et les principales applications existantes.

La seconde partie décrit précisément une carte à microprocesseur : la carte CQL[2] de chez Gemplus. En effet, notre application carte-santé fonctionne avec cette carte. Nous expliquons dans cette partie les principes de base qui régissent la carte; avec notamment sa structure en tables et son langage de requêtes de type SQL.

2. Les notions de base

2.1. Historique

Depuis le début des années 1970, la carte à mémoire connaît un succès croissant [Para86][GUV91][Grim92]. Elle remplace de plus en plus des techniques anciennes de stockage d'informations telles que les pistes magnétiques et le procédé d'embossage (écriture en relief sur la carte plastique). En effet, cette nouvelle technologie de la carte à microprocesseur laisse entrevoir la création d'un grand nombre d'applications. Ceci grâce au fait que la carte à microprocesseur est **un véritable ordinateur et est donc capable d'exécuter des programmes**. Elle se distingue fortement de ses prédécesseurs qui sont caractérisés par une passivité certaine. De plus, ces dernières présentent des inconvénients majeurs : effacement de l'inscription par un champ magnétique intense ou par la chaleur, faible capacité de stockage et reproduction possible suite à un vol.

En 1974, le français **Roland Moreno** a eu l'idée de remplacer l'enregistrement magnétique par un composant électronique. Il dépose un certain nombre de brevets. En 1978, une équipe de BULL CP8 invente le concept **de M.A.M. (Microprocesseur Autoprogrammable Monolithique)**. La première carte à microprocesseur sortira des ateliers de chez BULL en 1981.

[2] CQL est une marque déposée du groupe Gemplus

Mais la notion de carte à microprocesseur est bien souvent mal perçue. Levons immédiatement les ambiguïtés qui pourraient apparaître en définissant successivement les notions de carte à mémoire, de carte à microcircuit et de carte à microprocesseur, .

- La *carte à mémoire* désigne une carte qui permet uniquement le stockage d'informations. Les cartes magnétiques et les cartes laser entrent dans cette catégorie.

- La *carte à microcircuit* contient une composante électronique qui est dotée d'une logique câblée. Celle-ci est prédéterminée et le traitement est donc figé. Les télécartes françaises ou allemandes sont de bons exemples.

- La *carte à microprocesseur*, appelée également *carte à puce* ou *Smart-card* en anglais, comporte un microprocesseur. Ces cartes permettent un traitement évolué et personnalisable. Ce traitement est réalisé par l'intermédiare de programmes qui sont exécutés par la carte.

2.2. Les phases de vie d'une carte

- Phase 1 : fabrication de la carte à microprocesseur par un constructeur

C'est lors de cette première phase, qu'il faut tenir compte des contraintes du marché et du type d'application, pour déterminer quels types de processeur et de mémoires sont nécessaires. Il s'agit ensuite d'intégrer dans la ROM son système d'exploitation, appelé le masque. Enfin, on place dans la mémoire de données, à un endroit prévu à cet effet, le numéro de série de la carte.

- Phase 2 : personnalisation par l'émetteur de la carte en fonction de l'application

Les informations concernant l'identification et l'authentification sont insérées lors de cette phase. L'application peut également être personnalisée : paramétrage de certaines zones du système, réservation de certaines zones de travail et ajout de fonctionnalités.

- Phase 3 : distribution et utilisation de la carte

A partir du moment où un porteur ou un utilisateur a été déclaré, on considère que la carte entre dans sa phase d'utilisation. Elle quittera cet état si elle est remplie (dans le cas des mémoires EPROM), si elle est bloquée ou si elle est périmée. Une carte se bloque si l'utilisateur a effectué un certain nombre (typiquement trois) de présentations erronées. Une carte est périmée si une date de clôture d'utilisation avait été prévue et est atteinte.

- Phase 4 : mort de la carte

2.3. La description physique d'une carte

2.3.1. Les différents composants

Comme nous le montrons dans la *Figure 2*, une carte à microprocesseur est composée de trois éléments : **un microcircuit, une pastille de contacts, le support plastique**. L'opération qui consiste à coller le microcircuit et la pastille de contacts sur le support plastique s'appelle l'encartage.

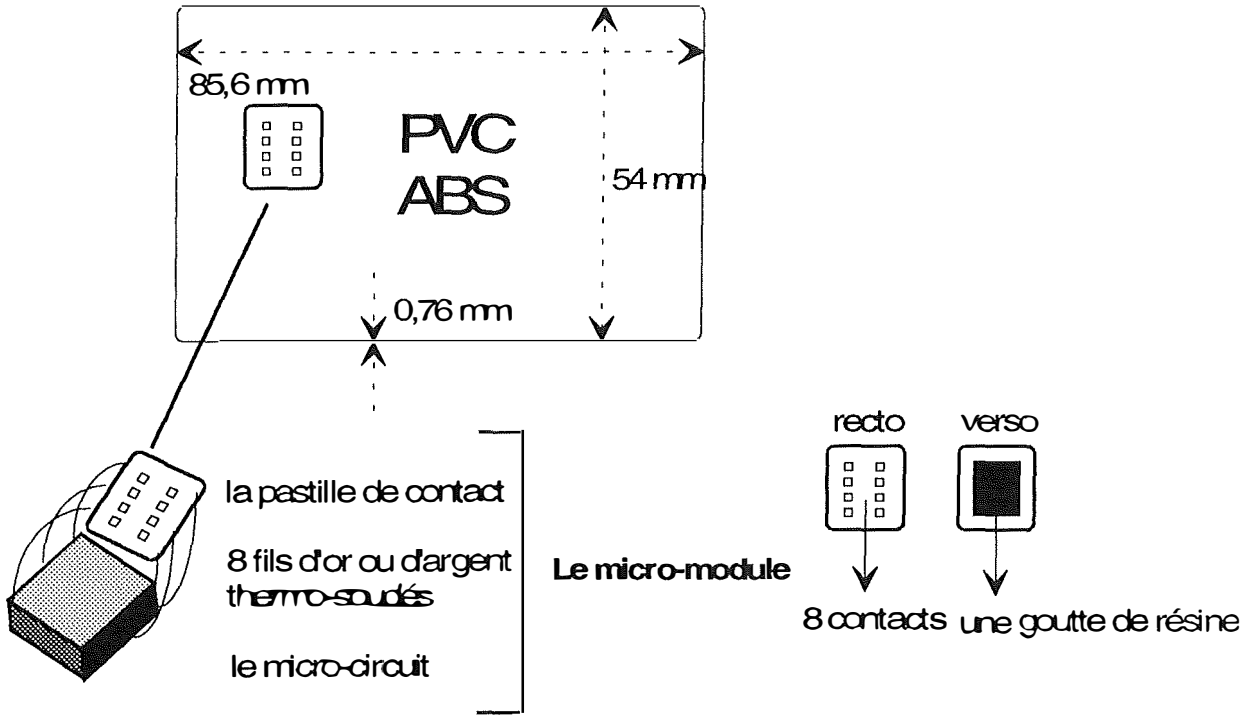


Figure 2 : Les caracteristiques physiques d'une carte à microprocesseur

Les échanges d'information entre la carte et le monde extérieur s'effectuent par l'intermédiaire des contacts. La position des contacts, leur dimension et leur fonction sont régies par la norme ISO 7816-2. Huit contacts sont prévus et seulement six sont utilisés. Nous explicitons les fonctions de ces 6 contacts dans le *Tableau 1*.

Dénomination :	Fonction :
VCC	alimentation électrique du circuit entre 3 et 5,25 volts
RESET	initialisation physique et logicielle
CLOCK	signal d'horloge : 3,58 ou 4,92 Mhz
GND (ground)	référence 0 volt
VPP	tension de programmation (12-15 V pour EPROM et 18-20 V pour EEPROM)
I/O	entrées et sorties (half-duplex)

Tableau 1 : Les fonctions des contacts

2.3.2. L'architecture d'un microcircuit

L'architecture d'une carte à microprocesseur est très comparable à celle d'un micro-ordinateur. Pour assurer une sécurité optimale, les constituants physiques du microcircuit sont intégrés sur un unique substrat de silicium :

- microprocesseur 8 bits
- mémoire RAM
- mémoire ROM
- mémoire EPROM ou EEPROM
- bloc de sécurité

Ces éléments sont **reliés par un seul bus**. L'intérêt d'une carte monocircuit réside dans l'absence de connexion directe entre l'extérieur et la mémoire. Tous les dialogues avec l'extérieur passent par le processeur qui effectue un contrôle sur toutes les requêtes. BULL, qui a été la première société, en 1979, à implémenter une telle architecture, a nommé cette carte un M.A.M. (Microcalculateur Autoprogrammable Monolithique) ou S.P.O.M. en anglais (Self-Programmable One-Chip Microcomputer). C'est un microcalculateur car il possède un microprocesseur, des mémoires de programmes et de travail. Il est autoprogrammable; en effet, le M.A.M. est capable d'écrire et de modifier le contenu de sa propre mémoire sans interrompre l'exécution du programme. Enfin, il est monolithique car le microcircuit est intégré dans un substrat unique.

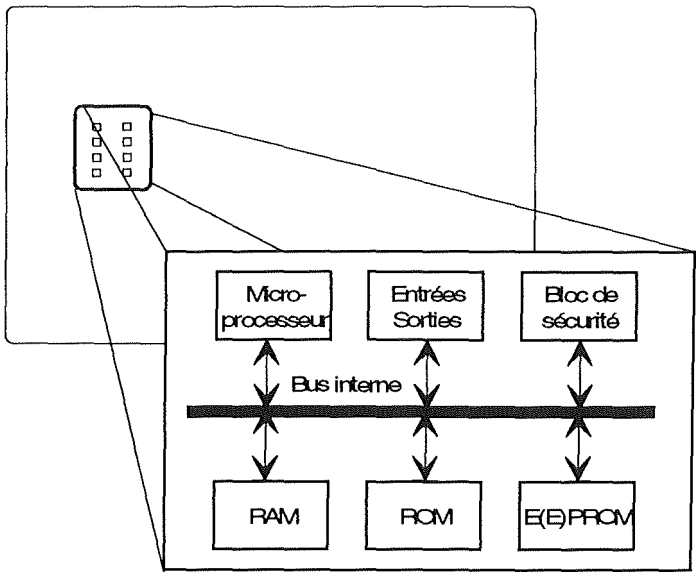


Figure 3 : L'architecture du micromodule

Détaillons quelque peu les différents éléments qui composent le microcircuit présenté à la *Figure 3*.

- La mémoire RAM (Random Access Memory):

C'est une mémoire de travail volatile qui est utilisée pour les calculs et les transferts de données. Elle est accessible uniquement par le M.A.M. et sa taille est limitée à **512 octets**.

- La mémoire ROM (Read-Only Memory) :

Celle-ci contient le système d'exploitation de la carte, appelé masque. Ce masque est inséré dans la carte lors de la fabrication du microcircuit. Sa taille est varie entre **10 et 12 Koctets**.

- La mémoire réservée aux données :

On retrouve deux grands types de mémoires. La mémoire EPROM (Electrically Programmable ROM) est une mémoire non volatile effaçable par des rayons ultraviolets. La mémoire EEPROM (Electrically Erasable and Programmable ROM) est non volatile effaçable électriquement et programmable par injection de charges électriques. La taille de ces mémoires est généralement comprise entre **8 et 10 Koctets**.

Quel que soit le type choisi, l'organisation de la mémoire est comparable. La mémoire est divisée en six zones ayant chacune leurs caractéristiques :

- la zone secrète contient les clés d'accès et un jeu secret, qui sont accessibles uniquement au microprocesseur;
- la zone mémoire d'états dans laquelle est inscrit l'historique des présentations de clés;
- la zone confidentielle dont l'accès est protégé par un code secret;
- la zone libre dont l'accès en lecture est libre;
- la zone de transaction ou d'application qui peut recevoir les inscriptions successives;

- la zone de fabrication qui contient les adresses des différentes zones et les caractéristiques de protection de chacune d'elles.

- Le bloc de sécurité :

Son rôle est d'assurer les contrôles relatifs à la sécurité physique du composant. C'est-à-dire protéger la carte de toute attaque extérieure. Pour cela, des mécanismes tels que la détection de lumière, la détection de la variation de la fréquence d'horloge et la détection de l'exposition à la chaleur sont mis en place.

Certains composants disposent également d'un circuit dénommé matrice de sécurité. Cette matrice permet de régir les accès aux différentes mémoires. La *Figure 4* présente un exemple de matrice de sécurité dans laquelle L signifie lecture, E écriture et X exécution.

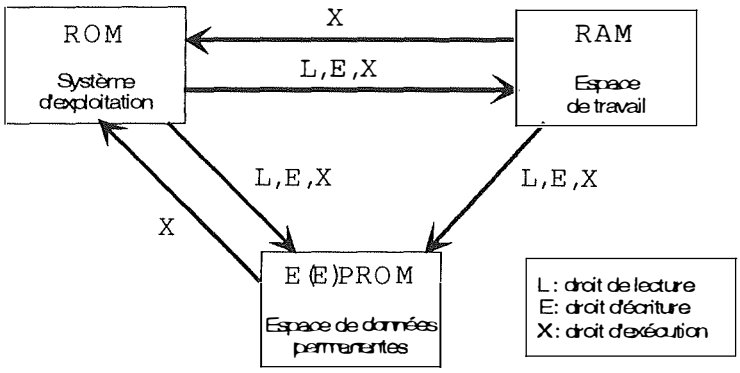


Figure 4 : La matrice de sécurité

2.3.3. Le protocole d'échange

a) Généralités

Le mode de transmission entre la carte et l'ordinateur hôte est **asynchrone half-duplex**. On ne dispose que d'une seule ligne pour les entrées/sorties. Cela implique l'utilisation du mode half-duplex. La norme ISO 7816-3 définit deux types de protocoles : T=0 et T=1. Le protocole T=0 permet la transmission d'octets, tandis que T=1 autorise le transfert de blocs.

La transmission d'un caractère est illustré par la *Figure 5*. Le bit de démarrage assure la synchronisation. La détection d'erreur est gérée grâce au bit de parité. Entre l'envoi de deux caractères, une séparation est prévue afin de permettre la signalisation d'une erreur dans la transmission.

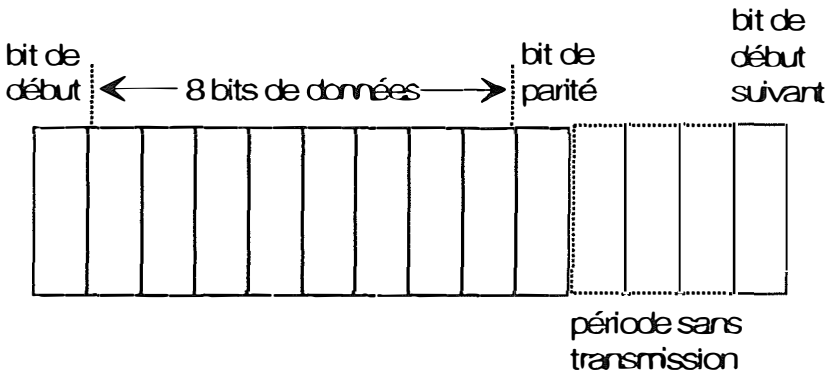


Figure 5 : La transmission d'un caractère

b) La réponse au reset

Une fois que la carte est mise sous tension, elle renvoie différents octets nécessaires au paramétrage de la transmission. Cet ensemble d'octets, appelé réponse au reset, contient cinq champs :

- TS : un octet pour la synchronisation et les conventions de codage pour les octets qui suivent.
- T0 : un octet de format qui signale les caractères optionnels qui sont effectivement présents dans la suite du message.
- TA, ..., TD des octets optionnels : ces octets définissent les caractéristiques physiques de la carte ainsi que les caractéristiques logiques du protocole d'échange.
  TA : définit l'horloge et la vitesse de transmission
  TB : définit les tensions nécessaires pour écriture dans la mémoire (VPP)
  TC : spécifie le temps de transmission d'un caractère.
  TD : spécifie le protocole d'échange (T=0 ou T=1).
- Les octets T1 ...T15 :
  Ces octets permettent d'identifier le type de carte, la référence de l'émetteur et l'état de la carte.

c) Le protocole T=0

L'une des caractéristiques de ce protocole est que la carte est esclave du système hôte. Seul le terminal envoie des ordres à la carte. Dans ce protocole, le lecteur initialise le dialogue par l'envoi d'une commande à la carte et attend une réponse. Ensuite, les commandes se succèdent. Etant donné que la communication est en half-duplex, les messages ne peuvent être envoyés que dans un seul sens à la fois. La coordination de la communication est implicite. La carte reçoit une commande du lecteur, la traite et renvoie les informations nécessaires à celui-ci. Trois types de commande sont possibles :

* soit elle contient des données à insérer dans la carte.
* soit elle demande des données à la carte.
* soit elle envoie des données à la carte et attend des données en retour. Dans ce dernier cas, le lecteur fait suivre sa première commande d'une deuxième de type get_response, afin d'obtenir une réponse.

La structure d'un message est la suivante :

CLA la classe d'instruction
INS le code de l'instruction
P1 le premier paramètre (par exemple une adresse mémoire)
P2 le second paramètre
P3 la longueur du bloc de données; si P3 est égal à zéro, soit on n'envoie aucune donnée à la carte, soit la commande est une demande d'informations à la carte.

Dans la *Figure 6*, nous montrons deux cas possibles de transmission. Dans le premier cas, une donnée est envoyée à la carte. Dans le second, il s'agit d'une requête à la carte. Quand la carte reçoit une commande, elle envoie un accusé de réception. Suite à l'envoi ou la réception de données, la carte fournira au lecteur deux octets de statut pour indiquer à celui-ci dans quelles conditions s'est déroulée l'exécution de la commande.

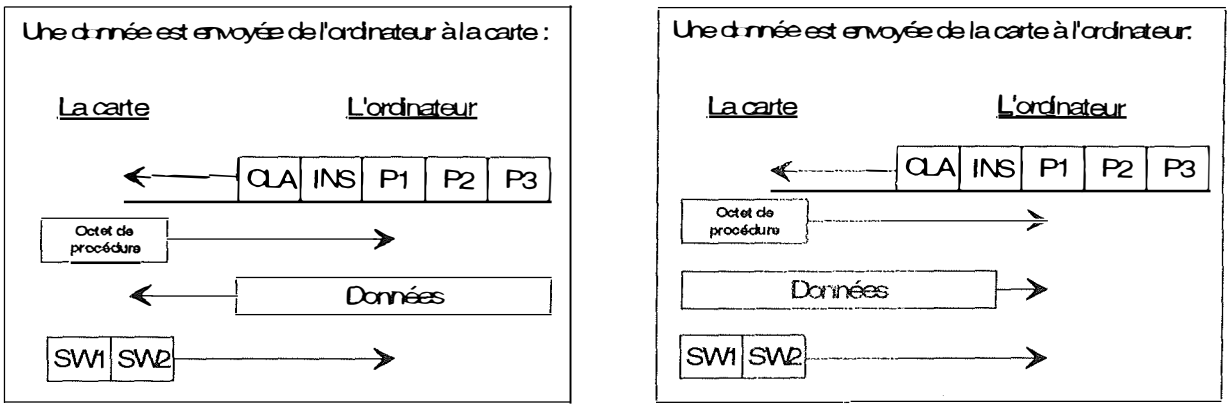


Figure 6 : L'échange avec le protocole T=0

d) Le protocole T=1

Le protocole T=1 gère la transmission asynchrone de blocs en half-duplex. Ce protocole peut être vu dans le monde OSI comme le niveau 2 : la couche liaison de données. Le protocole T=0 correspond lui à la couche physique. Le contrôle de flux, le chaînage de données et la détection des données sont les trois principaux apports du protocole T=1. Ces différents éléments de gestion supplémentaires entraînent inévitablement une charge plus importante pour la mémoire RAM de la carte. Celle-ci doit en permanence garder la dernière information envoyée au lecteur afin de pouvoir l'émettre à nouveau en cas de besoin.

La structure des blocs est illustrée dans le *Tableau 2* :

Champ de prologue			**Champ d'information**	**Champ d'épilogue**
Adresse	Octet de contrôle du protocole	Longueur	(optionnel)	Détection d'erreur
NAD	PCB	LEN	INF	EDC
1 octet	1 octet	1 octet	0 - 254 octets	1 ou 2 octets

Tableau 2 : La structure des blocs pour le protocole T=1

Le champ de prologue

L'octet NAD (node of address) représente l'adresse source et l'adresse du destinataire.
L'octet PCB (protocol of control byte) permet l'identification des trois types de blocs : bloc d'information, bloc de réception prête ou bien bloc de supervision.

Le bloc d'information est utilisé pour transmettre des commandes et des données du lecteur à la carte et vice-versa.

Le bloc de réception prête joue le rôle d'accusé de réception dans le cas du chaînage de blocs.

Le bloc de supervision établit des paramètres de contrôles tels que la resynchronisation. Il sert également d'accusé de réception dans le cas d'une transmission sans chaînage.

L'octet LEN indique le nombre de bytes dans le champ d'information. Il peut prendre une valeur entre 00 et FE h. Cela permet le transport d'informations d'une longueur maximale de 254 octets.

Le champ d'information contient les informations qui sont envoyées d'un interlocuteur à l'autre.

Le champ d'épilogue comprend le ou les octets nécessaires à la détection d'erreurs.

2.4. La sécurité dans les cartes à microprocesseur

2.4.1. La sécurité physique

Le microcircuit est un M.A.M. sur un seul bloc de silicium dont l'architecture assure d'elle-même une grande part de sécurité. La *Figure 7* rappelle cette architecture.

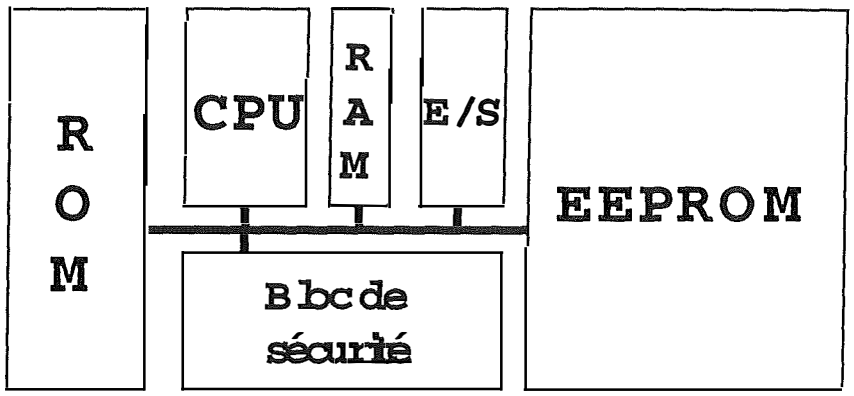


Figure 7 : L'architecture d'un micromodule

Le microprocesseur a **un fonctionnement tout à fait classique** : acquisition d'une instruction, décodage et exécution de celle-ci. Par contre, une différence importante au point de vue de la sécurité apparaît dans la programmation de la mémoire. Dans les microprocesseurs classiques, cette tâche est laissée à un dispositif extérieur.

Dans le cas du M.A.M., **c'est lui seul qui effectue cette programmation**. Pour cela, il dispose d'un ensemble d'instructions et d'une structure (illustrée à la *Figure 8*) qui réalisent ce travail. Il présente l'adresse d'écriture et l'adresse de la donnée à la mémoire, applique la tension de programmation, la retire et reprend le déroulement de son programme. Deux registres sont dédiés à ce travail, **Adr** et **Don** pour mémoriser respectivement la donnée et son adresse. Ces registres sont bloqués pendant la phase d'écriture par le signal E. L'activation du signal VEC (Validation d'ECriture) déclenche l'écriture dans la mémoire. Ces opérations sont commandées et contrôlées par l'unité centrale du microcircuit.

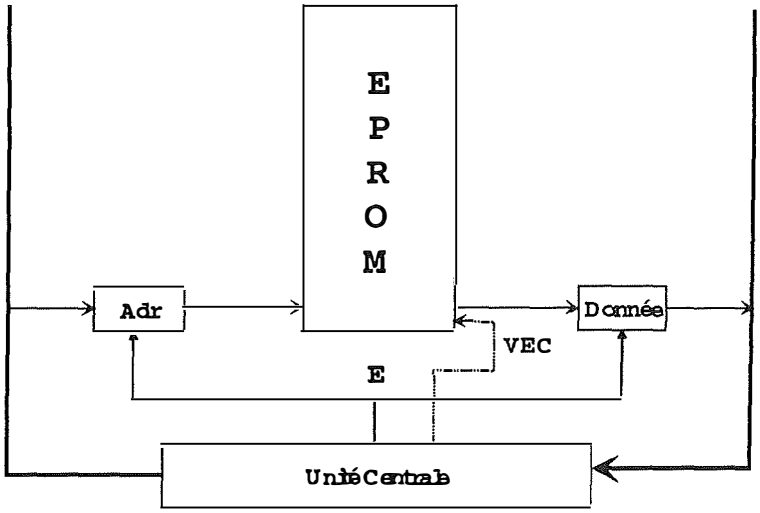


Figure 8 : Un enregistrement dans la mémoire

Détaillons quelque peu les menaces auxquelles la carte pourrait être confrontée :

- Analyser le module avec un microscope électronique pour lire le contenu de la ROM ou de l'EEPROM;
- Placer des sondes pour espionner les données transitant sur le bus ou en RAM;
- Provoquer des dysfonctionnements tels que variation de la température, de la tension ou de la fréquence.

La carte est pourvue des mécanismes de protection suivants :

- Afin de se protéger face au troisième type d'attaque mentionné dans le paragraphe précédent, la carte possède des détecteurs de lumière, de passivation, de température, de tension et de fréquence.
- Le seul bus présent dans la carte est littéralement enterré dans celle-ci. Il est impossible d'y accéder sans rompre le microcircuit.
- Les adresses ROM et EEPROM sont brouillées.
- La logique des circuits est tout à fait sauvage. Cela signifie que les constructeurs n'hésitent pas à ajouter des circuits qui ne servent à rien si ce n'est à brouiller les pistes des éventuels fraudeurs.
- Lors de la lecture en ROM et en EEPROM, un courant aléatoire, et donc connu uniquement de la carte, est utilisé pour les accès.
- Des matrices de sécurité physique sont mises en place (comme nous l'avons expliqué ci-dessus cfr. *Figure 4*) pour éviter un accès frauduleux aux différentes mémoires. Cette matrice interdit par exemple toute copie de la mémoire ROM.

2.4.2. La sécurité logique

La sécurité logique comprend la protection globale des informations. Il existe trois menaces principales :

- La lecture non autorisée d'informations protégées.

- L'altération d'informations du système ou d'une application.
- L'écriture d'informations sans y être autorisé.

Ici aussi, d'importants mécanismes ont été mis en place afin de protéger la carte contre ces types d'attaques.

- L'authentification est obligatoire pour accéder à certaines zones de la carte.
- La carte mémorise l'origine des informations qui sont écrites dans sa mémoire.
- Un contrôle de la cohérence des informations contenues dans la carte est effectué au moyen d'informations redondantes du type CRC ou Checksum.
- Un certain nombre de verrous permettent d'établir dans quelle phase de vie la carte se trouve.
- La possibilité de chiffrer des informations sensibles aussi bien au niveau de la transmission que du stockage.

2.5. Un survol des applications

Nous exposons dans cette partie les différents types d'applications qui sont et qui pourraient être concernés par la carte à microprocesseur [JJV95]. Nous allons également montrer qu'une application peut regrouper tous les types de services imaginables.

Nous pouvons dégager quatre domaines d'application qui sont les suivants :

1. le paiement
2. la sécurité physique d'accès
3. la sécurité logique d'accès
4. le dossier portable

Le premier domaine recouvre plusieurs aspects comme le pré-paiement, les cartes de crédit, de débit et le home banking. Le second et le troisième concernent la sécurité d'accès. Le dernier regroupe toutes les applications où schématiquement le porteur est « porteur de son dossier ». Notre exemple de carte santé entre bien entendu dans ce domaine d'application.

2.5.1. Les cartes de paiement

Il est intéressant de noter que ce domaine est **à l'origine de cette technologie**. Dès le dépôt des premiers brevets par Roland Moreno, **les industriels ont de suite senti l'enjeu** d'une telle innovation dans le domaine du paiement.

Le pré-paiement est une application appelée également porte-monnaie électronique. Le pré-paiement constitue à charger préalablement la carte avec des unités de consommation. Ces unités sont pré-payées. En général, la valeur chargée dans ces cartes est assez faible. Au cours de son utilisation, la carte se voit débitée du nombre d'unités consommées. Le plus souvent, une fois que la carte s'est vue débitée de l'ensemble des unités, elle est jetée. Deux fonctionnalités primordiales doivent être garanties : l'identification de l'émetteur et le décompte des unités consommées. L'identification est très importante pour l'émetteur car elle doit lui

garantir que le porte-monnaie électronique présenté par l'utilisateur soit valable. De même, la carte doit assurer que tout service sera décompté en terme d'unités d'utilisation.

L'exemple le plus célèbre en France est la télécarte qui permet d'utiliser le téléphone public. Dans ce cas, il s'agit de cartes à logique câblée qui peuvent contenir de 40 à 80 unités téléphoniques. En Belgique, la carte Proton a été testée dans le courant de l'année dernière à Wavre. Elle représente un véritable porte-monnaie électronique qui permet les petits achats de la vie courante. Depuis le mois de mai, elle est disponible dans plusieurs grandes villes belges dont Namur.

Les cartes à microprocesseur apportent des solutions aux problèmes posés par l'utilisation des cartes magnétiques (sécurité, usure, ...) et génèrent avec leurs capacités fonctionnelles de nouveaux services aux systèmes de paiement. Les cartes de paiement sont les cartes les plus répandues dans le monde. Actuellement, les cartes utilisées par les banques sont des cartes magnétiques. Un alliance des deux géants américains Visa et Mastercard donnera prochainement lieu au remplacement de ces cartes par des cartes à microprocesseur. On imagine très bien les répercussions économiques et industrielles d'une telle décision.

En France, les banques ont déjà adopté les cartes à microprocesseur depuis quelques années (la fameuse carte bleue). Elles offrent les fonctionnalités suivantes :

- l'identification du porteur et de l'émetteur
- l'authentification de la carte et du terminal
- la signature électronique
- la gestion en ligne

Le principal avantage reconnu aux cartes à microprocesseur dans le domaine bancaire, est de permettre une forte diminution de la fraude. En effet, un simple investissement de 120.000 Frs permettrait à quiconque de dupliquer des cartes magnétiques. Par contre, pour envisager une fraude avec les cartes à microprocesseur, il faut investir une somme représentant plusieurs millions. La rapidité et la facilité d'utilisation sont les deux autres avantages les plus souvent cités en matière de carte bancaire à microprocesseur.

2.5.2. La sécurité physique d'accès

C'est une utilisation de la carte que l'on peut **assimiler à une clé d'ouverture** d'un lieu. Ainsi dans certaines entreprises, quand un membre du personnel se déplace dans l'établissement, il doit présenter à un portier électronique sa carte. L'authentification peut se dérouler par l'intermédiaire d'un mot de passe ou par des techniques plus compliquées telles que la biométrie, la reconnaissance du fond de l'œil, les empreintes digitales.

Ces applications utilisent des supports d'identification. On utilise ce système pour sécuriser des locaux d'entreprises où une haute sécurité d'accès est nécessaire. L'identification présente sur la carte est utilisée par le portier pour accorder ou non l'ouverture de la porte, suite à une comparaison avec la valeur de référence contenue dans la carte. La mémorisation de données sur la carte est possible et peut par exemple servir les traces du passage d'un employé.

2.5.3. La sécurité logique d'accès

La connexion à un serveur ou à un site distant pose un certain nombre de problèmes. L'identification et l'authentification réciproques (de l'appelant et de l'appelé) sont nécessaires. Des techniques de chiffrement seront peut-être utilisées pour toutes les communications. Dans ce cas, la clé de chiffrement pourrait se trouver dans la carte. Le système distant doit pouvoir écrire à distance sur la carte. On parle alors de téléécriture. Lors de celle-ci, on cherchera à s'assurer que l'écriture s'est bien déroulée en la contrôlant au moyen d'une technique de certification.

On trouve un grand nombre d'applications qui utilisent les techniques décrites ci-dessus. Nous pensons notamment à la protection de la messagerie électronique au moyen de signature électronique et de chiffrement. Le renforcement de la sécurité lors de transactions sensibles peut également être facilité par l'entremise de ces techniques.

Le GSM (Groupe Spécial Mobile) a défini un réseau unique de téléphones cellulaires, en utilisant la carte à microprocesseur comme module d'identification d'abonné, appelé carte SIM (Subscriber Identity Module).

2.5.4. Les dossiers portables

Un dossier portable est **un support carte à mémoire pour un dossier d'informations concernant le porteur du dossier**. Les informations peuvent être par exemple le dossier scolaire, universitaire ou santé du porteur. Les informations contenues dans ce dossier sont utilisées par des prestataires de services comme les administrations des universités ou les médecins.

La mémorisation de données est la fonction la plus importante de la carte dans ce cas. Elle devra pouvoir offrir une grande capacité de stockage et également le possibilité d'effacer les données obsolètes. Il faut également apporter une grande importance à la protection des données. Il est intéressant qu'il existe des degrés d'accès aux données stockées.

Passons en revue quelques dossiers portables existants.

- Une carte d'assurance maladie :

En Allemagne, une carte d'assurance santé a été développée et distribuée à 78 millions de personnes. En Espagne, SANIRED est un projet d'assurance médicale privée.

- Une carte dossier médical portable :

Le projet pilote de carte santé à Rimouski (Québec) a vu le jour il y a quelques années et continue à se développer. Il s'agit d'un dossier médical d'un patient contenant les

ordonnances pour le pharmacien, les dernières consultations, des informations d'urgence (telles que les allergies, etc.) et même des résultats d'examens.

• Une carte de médecine ambulatoire :

C'est une carte de suivi de soins qui permet le transfert des informations entre des équipements installés à domicile ou utilisés en déplacement et les centres hospitaliers.

• Un carte de dialyse :

L'HEMACARD a été réalisée à Namur en collaboration avec les médecins de l'hôpital de Mont-Godinne [NgBoSa94]. L'équivalent en France porte le nom de carte DIALYBRE.

3. La carte CQL[3]

3.1. Introduction

La carte CQL (Card Query Language) est un produit Gemplus [Gem93]. Elle fut conçue en 1992 par RD2P . Cette carte possède comme particularité d'être interrogeable par l'intermédiaire d'un langage de requêtes de bases de données. Ce langage CQL (Card Query Language) est un sous-ensemble du Structured Query Language (SQL), le langage de gestion de bases de données relationnelles le plus usité [ISO94].

La carte CQL est multi-applicative. On peut y stocker des données concernant différentes applications. Plusieurs utilisateurs peuvent y être créés et ce pendant toute la durée de vie de la carte.

Cette carte permet au programmeur de ne pas se préoccuper des accès aux données. Tout ce qui concerne l'organisation et la gestion des données est géré par le système d'exploitation de la carte.

[3] CQL est une marque déposée du groupe Gemplus

3.2. Principes de base

3.2.1. Les tables, les vues et les dictionnaires

La table est l'objet central de cette carte. En effet, **toutes les informations liées au porteur sont stockées dans des tables**. La gestion et la sécurisation de ces données sont assurées par des tables systèmes. Ces tables systèmes sont au nombre de quatre :

- La table répertoire de toutes les tables et les vues stockées dans la carte, composée des colonnes suivantes :
  - le nom de la table ou de la vue
  - le nom du propriétaire
  - le type de l'objet (T pour une table et V pour une vue)
  - le niveau de sécurité
  - le nombre de colonnes
  - s'il s'agit d'une table, le nom des colonnes
  - s'il s'agit d'une vue, des informations exigées par le système

- La table des intervenants contenant les identifiants et les ratifications, composée des colonnes suivantes :
  - le nom de l'utilisateur
  - le profil de l'utilisateur (A pour gestionnaire d'application, E pour émetteur et U simple utilisateur).
  - le créateur de l'utilisateur
  - le taux de ratification
  - le compteur de ratification

- La table des privilèges regroupant les autorisations d'accès aux objets
  - le nom de l'objet
  - le nom de l'utilisateur
  - le privilège
  - le créateur de l'objet

- La table des clés qui contient les mots de passe et les clés de chiffrement

La composition de ces tables implique que tout objet de la base de données doit posséder un nom unique. On ne pourra donc jamais avoir deux tables de même nom ni deux utilisateurs de même nom.

Pour une question d'optimalisation de l'espace des données, la structure dynamique de liste a été adoptée. C'est ainsi que la carte contient une liste de tables, que chaque table est composée d'une liste de lignes et qu'une ligne est quant à elle une liste de colonnes.

En plus des tables, le langage CQL permet le déclaration de vues sur une table. Une vue permet un accès restreint à une table. On peut de cette façon définir une granularité d'accès différente pour tous les utilisateurs.

En CQL, il existe également la notion de dictionnaire. Tout utilisateur peut se créer un dictionnaire. Celui-ci est une vue sur les différentes tables systèmes avec une condition de sélection correspondant au propriétaire. Le contenu de cette vue est ainsi limité aux seuls objets appartenant à cet utilisateur. De cette façon, les dictionnaires permettent à leur créateur d'avoir un aperçu de toutes les tables, vues et utilisateurs créés par leur soin.

Le contenu de ces dictionnaires est maintenu automatiquement par le système. Le créateur du dictionnaire peut permettre la consultation de celui-ci par d'autres utilisateurs.

3.3. Les utilisateurs

Le SGBD (Système de Gestion de Base de Données) est le système d'exploitation de la carte. Il gère les objets et les intervenants. La gestion de ces derniers est similaire aux SGBD traditionnels : l'administrateur crée des intervenants et leur attribue des mots de passe. Une fois que l'utilisateur a entré un mot de passe correct, il est connecté au système et une session est ouverte. Durant cette session, tous les accès de l'intervenant à la carte sont contrôlés par le masque. Le SGBD possède quatre profils d'intervenants prédéfinis qui sont décrits dans le *Tableau 3* :

Profils	Intervenants	Fonctions
Public	l'utilisateur public, n'importe qui trouverait la carte	aucune création possible
SU	l'utilisateur standard (Standard User)	aucune création possible
AM	le gestionnaire d'application (Application Manager)	il peut créer des intervenants simples et des objets
CI	l'émetteur de la carte (Card Issuer)	il peut créer des gestionnaires d'application, des simples intervenants et des objets

Tableau 3 :Les types d'utilisateurs de la carte CQL

D'une manière générale, **l'émetteur est celui qui crée les applications sur la carte. Le gestionnaire d'application est celui qui est habilité à créer des tables et des utilisateurs de ces tables. Enfin, l'utilisateur est celui qui peut consulter, modifier, insérer ou supprimer des tables suivant les privilèges qui lui auront été reconnus.**

L'utilisateur Public existe sur toutes les cartes CQL et ne peut pas être supprimé. C'est un utilisateur comme les autres hormis le fait qu'il ne doit pas entrer de mot de passe lors de la présentation. Tout le monde a donc la possibilité de se présenter comme utilisateur Public.

Cette fonctionnalité est très utile pour permettre à n'importe qui de découvrir à qui appartient une carte égarée. Il suffit pour cela de prévoir une table avec des données administratives et de permettre à l'utilisateur Public d'y accéder.

Il peut y avoir "n" simples utilisateurs, "n" gestionnaires d'application mais un seul émetteur. C'est le seul intervenant qui a le droit de créer des gestionnaires d'application. Il peut être vu comme un super-utilisateur. Son nom et son mot de passe sont définis lors de la personnalisation de la carte.

A chaque utilisateur, sauf l'utilisateur Public, est associé, lors de sa création, un mot de passe et la plupart du temps un seuil de ratification *t*. Celui-ci signifie que si l'utilisateur présente un mot de passe erroné à *t* reprises, il ne pourra plus accéder aux données le concernant. A ce moment, seul le créateur de l'utilisateur en question pourra débloquer l'utilisateur.

Exemple :

Création de l'utilisateur CHRU :

```
create user CHRU  3 'mot de passe' ;
```

Présentation en tant qu'utilisateur CHRU :

```
present CHRU 'mot de passe';
```

Présentation en tant qu'utilisateur Public :

```
present PUBLIC;
```

Ce mécanisme de protection par un mot de passe est également utilisé pour l'émetteur. Dans ce cas, c'est lors de la personnalisation que le mot de passe et le taux de ratification auront été fixés. Si l'émetteur dépasse le taux de ratification, la carte est bloquée et seul le fabriquant pourra la débloquer.

3.3.1. Les éléments syntaxiques du langage CQL

a) Les types d'expressions

Il existe quatre types d'éléments syntaxiques : les caractères, les identificateurs, les chaînes de caractères et les opérateurs de comparaison. A partir de ceux-ci, on peut construire des prédicats et des conditions de recherches.

- Les caractères :

Le caractère est l'unité de base de tout élément syntaxique. Il est représenté par un octet. On retrouve trois catégories de caractères : les lettres (de a à z et A à Z), les chiffres (de 0 à 9) et un certain nombre de caractères spéciaux.

- Les identificateurs :

Les identificateurs sont utilisés pour identifier les tableaux, les vues, les utilisateurs et les colonnes. Ils sont composés de lettres, de chiffres et du caractère spécial '_'. Ils sont sujets à trois contraintes. Premièrement, leur longueur maximale est de six caractères. Ensuite, toutes les lettres d'un identifiant doivent être des majuscules. Enfin, le premier caractère de l'identificateur doit être une lettre.

- Les chaînes de caractères :

Le contenu d'une cellule des tables de la carte est composé uniquement de chaînes de caractères. La chaîne de caractères est donc le seul type de données reconnu par le langage CQL. Cela comporte certains désavantages comme le fait que l'on ne puisse effectuer une opération arithmétique sur une donnée immédiatement dans la carte. Il est toutefois possible d'effectuer ce traitement en-dehors de la carte ; ce qui finalement ne représente pas un gros inconvénient.

b) Les conditions de recherche

Une condition de recherche est composée d'un certain nombre de prédicats (cfr. *Figure 9 : Un prédicat*). C'est une expression de condition logique.

Les prédicats représentent des tests sur le contenu des cellules d'une table. Le résultat d'un prédicat est un booléen (vrai ou faux). Il est composé de trois éléments : le nom du champ de comparaison, l'opérateur de comparaison et la chaîne de caractères.

Exemple : `AGE > '20'`

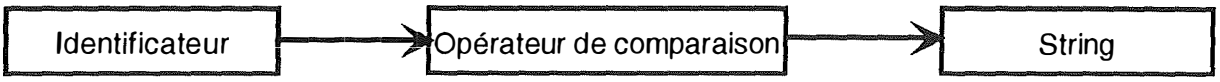


Figure 9 : Un prédicat

Six opérateurs de comparaison sont reconnus par la carte et décrits dans le *Tableau 4*:

Type :	Représentation :
l'égalité	=
l'infériorité	<
la supériorité	>
la différence	#
plus grand ou égal	G
plus petit ou égal	L

Tableau 4 : Les opérateurs de comparaison

Une condition de recherche est composée de un ou plusieurs prédicats sur une même table, comme le montre la *Figure 10*. Le résultat d'une condition de recherche est un booléen. Les prédicats qui composent la condition de recherche sont reliés par des opérateurs logiques 'AND'. L'opérateur 'AND' est le seul à pouvoir être utilisé. Les prédicats sont évalués séquentiellement et le système s'arrête à la première fausse condition. En théorie, le nombre de prédicats d'une condition de recherche est illimité. Toutefois, le tampon de la carte nous impose une longueur de 18 octets (les 'AND' n'étant pas compris) pour une condition de recherche.

Exemple :

```
AGE > '20' AND NOM ='DUPONT'
```

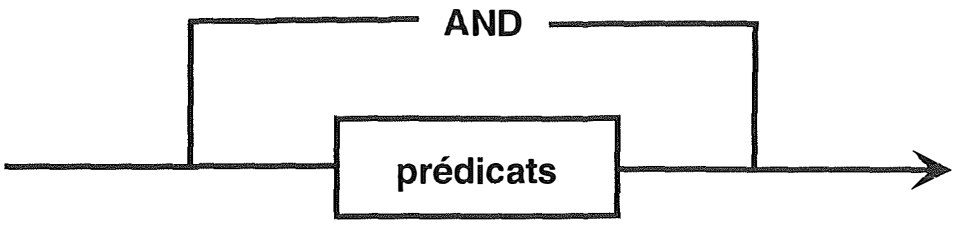


Figure 10 : Une condition de recherche

3.3.2. La transmission des requêtes à la carte

Lors d'un transfert de données entre la carte et le lecteur, les messages sont stockés dans un tampon dont la taille est de 64 octets. **La taille des messages envoyés à la carte est, de ce fait, limitée à la taille du tampon**. De plus, le contenu d'une cellule ne peut être modifié que par un seul bloc de données. Cela signifie qu'il n'est pas possible d'assigner une première valeur à une cellule et d'ensuite ajouter à cette cellule un autre bloc de données. La taille d'un message varie en fonction du type de la commande et des données qui l'accompagnent.

La forme des messages est définie en fonction de la norme ISO 7816 partie 3 et est illustrée par la *Figure 11*.

CLA	INS	P1	P2	LG	DATA

Figure 11 : Un message

CLA représente la classe du message. Le type d'instruction CQL est placé dans INS. P1 et P2 sont deux paramètres. LG indique la longueur des données. DATA représente les données qui sont le plus souvent une requête CQL encodée.

Le principe de base qui régit l'encodage des données est que tout identificateur, toute chaîne de caractères et tout nombre de colonnes sont précédés de leur longueur codée sur un octet. Cela permet une analyse déterministe du message. C'est ainsi que quand le système analyse un identificateur, il commencera par vérifier si la longueur annoncée correspond à la longueur réelle. Si ce n'est pas le cas, il s'arrête là et renvoie un message d'erreur.

Les commandes CQL et les nombres sont représentés sur deux octets. Le premier vaut toujours 01h et le deuxième est la représentation hexadécimale de la commande ou la valeur du nombre (entre 0 et 255). Une sélection ou la déclaration d'un curseur peuvent comprendre plusieurs éléments de comparaison. Il s'agit donc de faire précéder les comparaisons de leur quantité, qui est codée sur deux octets de la même façon qu'un nombre.

Exemple :

Le codage du nombre de comparaisons dans la condition de recherche AGE < 50 and SEXE = 'F' est le suivant
01h 02h

Voici le codage associé à la création de la table T_1 composée de trois colonnes.

create table T_1 (name,firstn,date);

create table	**01h** 54h
T_1	**03h** 54h 5Fh 31h
le nombre de colonnes	**01h** 03h
NAME	**04h** 4Eh 41h 4Dh 45h
FIRSTN	**06h** 46h 49h 52h 53h 4Eh
DATE	**04h** 44h 41h 54h 45h

Les premiers octets de chaque élément représentent la longueur de l'élément qui suit. Le message entité aura donc la forme présentée dans la *Figure 12* :

CLA	INS	P1	P2	LG	DATA
00	01 54	00	00	12	**03** 54 5F 31 **01** 03 **04** 4E 41 4D 45 **06** 46 49 52 53 4E **04** 44 41 54 45

Figure 12 : Un exemple de message transmis à la carte

3.3.3. Les principales commandes CQL

Le *Tableau 5* présente les trois catégories de commandes. Dans les paragraphes suivants, nous nous contenterons de détailler les commandes les plus utilisées. La description de toutes les commandes se trouve en annexe.

Gestion des utilisateurs	Définition des structures et des accès aux données	Manipulation des données
create application	create table	declare cursor
create user	create view	open
present	create dictionary	fetch
change password	drop table	fetch next
unlock	drop view	insert
delete user	grant	erase
create key	revoke	updatec
status		begin transaction
authenticate		commit
check		rollback
		read record

Tableau 5 : Les commandes CQL

a) Les commandes de gestion des utilisateurs

CREATE APPLICATION

CREATE APPLICATION <nom du gestionnaire> <taux de ratification> <mot de passe>;

< nom du gestionnaire> : un identificateur
<taux de ratification> : un nombre

Cette commande définit un profil de gestionnaire d'application. Le nom et le mot de passe du gestionnaire d'application devront être introduits lors de l'ouverture d'une session. Le taux de ratification représente le nombre de fois que le gestionnaire peut se présenter avec un mot de passe erroné avant que la carte ne le lui interdise. Si le taux de ratification est nul, cela signifie qu'il n'y a pas de limitation.

CREATE USER

CREATE USER <nom d'utilisateur> <taux de ratification> <mot de passe>;

<nom d'utilisateur> : un identificateur

Cette commande permet de déclarer un utilisateur et est réservée à l'émetteur et aux gestionnaires d'application.

PRESENT

PRESENT <nom d'utilisateur> <mot de passe>;
PRESENT PUBLIC;

La commande PRESENT peut être utilisée par n'importe quel utilisateur afin d'ouvrir une session. Pour que celle-ci soit ouverte, il faut que le mot de passe soit valide et que l'utilisateur ne soit pas bloqué (suite à un dépassement de son taux de ratification). Une ouverture de session signifie qu'un contexte est créé en fonction du type de l'utilisateur.

Chaque présentation erronée augmente le compteur de ratification d'une unité. Quand celui-ci atteint le taux de ratification prévu à la création de l'utilisateur, l'utilisateur est bloqué. Le créateur de cet utilisateur est le seul habilité à débloquer l'utilisateur. Par contre, le compteur de ratification est remis à zéro quand l'utilisateur se présente avec succès.

La commande PRESENT PUBLIC ne nécessite aucun mot de passe, étant donné que son accès à la carte sera restreint.

b) Les commandes de structures et d'accès aux données

CREATE TABLE

CREATE TABLE <nom de la table> (<liste des colonnes>);

<nom de la table> : identificateur
<liste des colonnes> : < identificateur >[,< identificateur >...]

Exemple :
create table ETAT_C(NOM,PRE,NAIS,NAT);

GRANT ET REVOKE

GRANT <liste de privilèges> ON <nom de table, vue ou dictionnaire> TO <nom d'utilisateur>;
REVOKE <liste de privilèges> ON <nom de table, vue ou dictionnaire> TO <nom d'utilisateur>;

<privilège> : insert | update | delete | select | all
<liste de privilèges> : <privilège>[,<privilège>]

La commande GRANT permet d'attribuer des privilèges à un utilisateur sur un objet. Par contre, la commande REVOKE retire des privilèges à un utilisateur.

c) Les commandes de manipulation de données

SELECT

SELECT * | <liste de colonnes> FROM <nom de l'objet> [where <condition de recherche>];

<liste de colonnes> : identificateur [,identificateur ...]
<nom de l'objet> : identificateur d'une table, d'une vue ou d'un dictionnaire

La commande SELECT est utilisée pour accéder à un sous-ensemble d'une table, d'une vue ou d'un dictionnaire. Les colonnes sélectionnées doivent appartenir à la table en question et il est autorisé de sélectionner plusieurs fois une même colonne. De plus, l'ordre dans lequel on place les colonnes sélectionnées ne doit pas nécessairement être le même que l'ordre défini dans l'objet.

SELECT est la commande de sélection la plus naturelle dans le langage SQL. Elle n'est toutefois pas implémentée dans la carte CQL. Ce qui ne signifie pas que l'on ne puisse pas l'utiliser. En effet, cette instruction sera décomposée avant d'accéder à la carte en instructions plus élémentaires (DECLARE CURSOR, FETCH, NEXT, ...).

INSERT

INSERT INTO <nom de table> VALUES (<liste de chaînes de caractères>);

<liste de chaînes de caractères> : chaîne de caractères [,chaîne de caractères ...]

Cette commande est utilisée pour insérer une ligne dans une table. L'utilisateur qui exécute cette commande doit posséder le privilège. Comme il s'agit d'insérer une ligne entière d'une table, toutes les colonnes doivent être présentes dans la liste des chaînes de caractères. La taille du tampon de la carte (64 octets) limite l'insertion des données à cette taille. Il convient alors de faire une insertion avec certaines colonnes vides (") et d'effectuer une mise à jour par la suite; comme l'illustre l'exemple suivant.

Exemple :

```
insert into TABLE1 values ('undeuxtrois','quatrecinq','sixsepthuit','');
updatec set COL4='neufdixonze';
```

UPDATEC

UPDATEC SET <nom de colonne> = 'chaîne de caractères' [,<nom de colonne> = 'chaîne de caractères'];

3.4. Les mécanismes de sécurité de la carte CQL

Le premier élément de sécurité est introduit par la gestion du masque. Celui-ci est le seul habilité à gérer les tables systèmes; en effet, il en est le propriétaire. Aucun autre utilisateur n'aura donc accès direct à ces tables. Le masque est composé de plusieurs couches. La *Figure 13* illustre ce fonctionnement. Toute exécution d'une requête CQL nécessite l'application successive de ces cinq étapes :

- Acquisition d'un ordre CQL : vérification syntaxique de l'ordre.

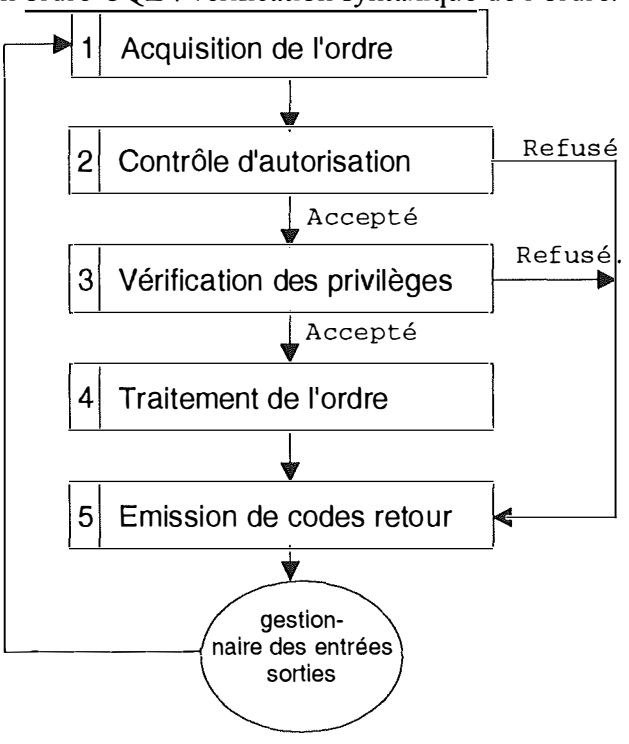


Figure 13 : L'exécution d'une requête CQL

- Contrôle d'autorisation : l'intervenant doit être habilité à l'exécution de l'ordre demandé.
- Vérification des privilèges pour la ou les tables concernées.
- Exécution de l'ordre : l'intervenant ayant passé avec succès les étapes de sécurisation, la requête est accomplie.
- Emission de codes retour : ces codes explicitent soit la raison du refus d'exécution d'un ordre soit des problèmes rencontrés lors de l'exécution.

Examinons à présent la sécurité du point de vue de l'utilisateur. Trois principes ont été pris en considération et découlent de la multi-applicativité de la carte CQL :

* les données entre deux utilisateurs différents doivent être strictement séparées.
* un utilisateur doit pouvoir accorder des privilèges sur les objets qu'il possède.
* la sécurité doit être dynamique. En effet, la durée de vie de la carte peut être relativement longue et donc les données doivent pouvoir être modifiées.

Le schéma de sécurité de la carte provient de ces principes et obéit aux deux règles suivantes :

- Règle 1 : l'intervenant (un émetteur ou un gestionnaire d'application) qui a créé un objet (une application, un utilisateur ou une table) en est **l'unique propriétaire** et lui seul peut effectuer des traitements sur cet objet.
- Règle 2 : le propriétaire d'un objet peut **accorder ou retirer des privilèges à un intervenant**. Les privilèges sont non transmissibles. Les différents privilèges sont la consultation, l'ajout, la suppression, ou la modification de données.

Nous illustrons ces deux règles par un exemple (cfr. *Figure 14*). Un émetteur E crée deux gestionnaires d'application App1 et App2. Le gestionnaire App1 crée les tables 1 et 2 ainsi que les utilisateurs 1 et 2. Le gestionnaire App2 crée les tables A et B et l'utilisateur 3. La *Figure 14* illustre cet exemple. L'émetteur ne peut accéder aux objets tables 1 et 2 (resp. A et B) créés par les gestionnaires d'application App1 (resp. App2). Le gestionnaire App1 n'a aucun accès aux objets générés par App2 (tables A et B). Par contre, le gestionnaire App2 s'est vu explicitement attribuer un accès à la table 2 par le gestionnaire App1. Cela vérifie donc bien que seul le propriétaire a le droit d'attribuer des droits d'accès aux objets qu'il a créé.

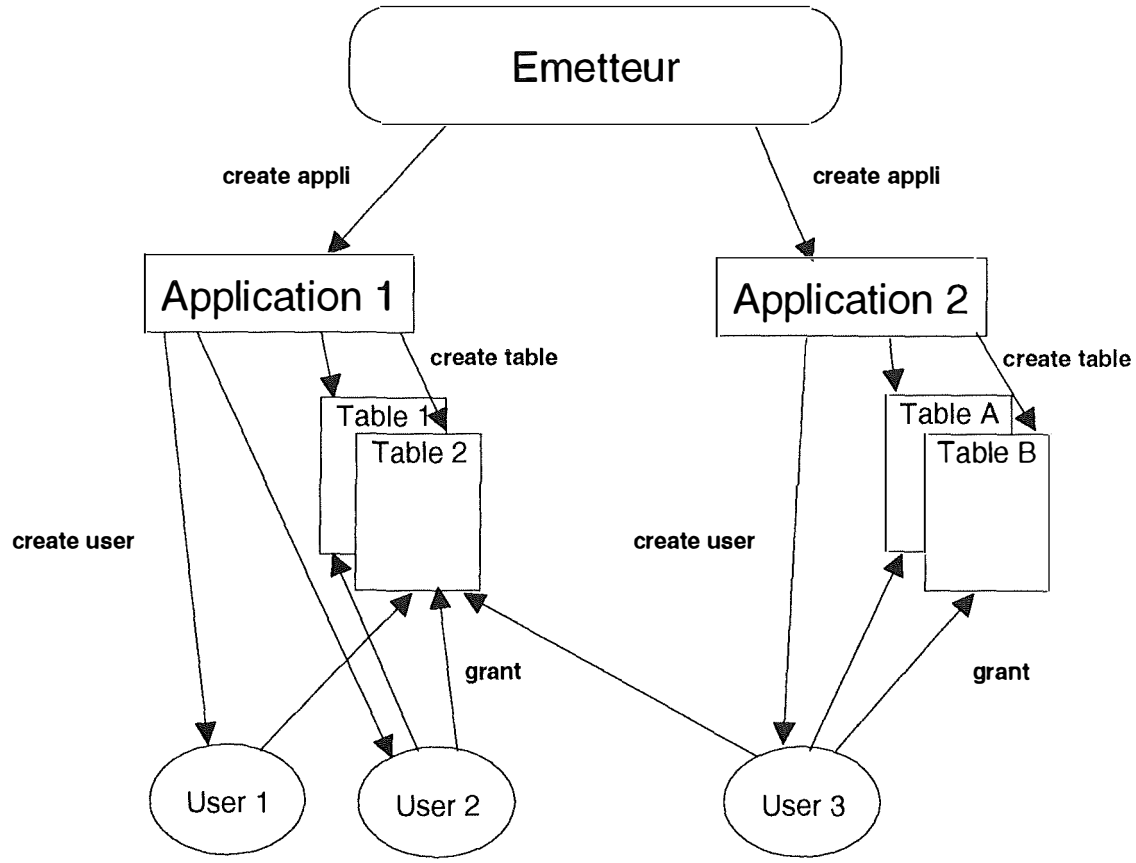


Figure 14 : Un exemple de structure supportée par la carte

3.5. Les limitations de la carte

La carte utilise le microprocesseur 8 bits Hitachi H8-310. Celui-ci a été choisi car il était un des premiers à disposer d'instructions d'écriture en EEPROM. La carte se caractérise donc par :

- une mémoire ROM de 10K octets
- une mémoire RAM de 256 octets
- une mémoire EEPROM de 8K octets

Cette architecture présente néanmoins un certain nombre de limitations qui toutefois ne paraissent pas être un handicap important dans le domaine de la carte à microprocesseur. Citons par exemple :

- le nombre maximum de colonnes dans une table ou dans une vue est de 10.
- le nombre de tables ou de vues sera limité par la taille de la mémoire EEPROM.
- la longueur des chaînes de caractères est également limitée : un élément d'une table aura une longueur maximale de 62 caractères, le nombre maximum de caractères qui composent un mot de passe est de 8 et il en est de même pour les clés.
- les identificateurs ont une longueur inférieure ou égale à 6 caractères; excepté les identificateurs de dictionnaires qui ne peuvent compter que 4 caractères.
- dans un declare cursor, on ne peut sélectionner que 5 colonnes.

Par rapport au langage de référence, le SQL, on ne pourra pas utiliser de types différents; le seul reconnu par la carte CQL est la chaîne de caractères. La jointure entre différentes tables est également proscrite. Cela entraîne une augmentation de la complexité du programme qui traite les données.

4. Conclusions

Ce chapitre nous a permis de découvrir un premier domaine concerné par notre application : la carte à microprocesseur. Nous sommes maintenant conscients **des avantages et des inconvénients** que présente un tel objet. D'une part, la carte à microporcesseur bénéficie **d'une sécurité importante**. D'autre part, ses possibilités physiques sont relativement faibles. Le problème essentiel reste **le peu de place mémoire disponible**.

Notre application est notamment utile pour montrer qu'il est possible de **contourner le problème de l'espace mémoire relativement réduit** dont dispose la carte à microprocesseur. Pour ce faire, nous utiliserons le monde de l'Internet qui vous est présenté dans le chapitre suivant.

Chapitre III :
Le monde Web

1. Introduction

Notre application a pour objectif d'être manipulable par l'entremise d'un browser standard. Tous les changements apportés au contenu de la carte passent par ce browser. Dans ce chapitre, nous apportons quelques précisions concernant tout ce qui touche à ce domaine. Dans un premier temps, nous expliquons d'où proviennent ces fameux termes : Web et Internet. Quelles sont les différences entre eux deux ? Quels en sont les fonctionnements ? Voici le type de questions auxquelles nous répondons.

Ensuite, nous présentons d'une façon détaillée le langage HTML (HyperText Markup Language). C'est celui-ci qui est utilisé pour créer des pages dans le monde Web. La dernière partie traite des CGI (Common Gateway Interface) qui sont les programmes exécutables sur un serveur. Ces programmes permettent au serveur de traiter des requêtes.

Le monde du Web est occupé à défrayer la chronique. La presse écrite n'en finit plus d'écrire des dossiers spéciaux sur ce phénomène. Les cybercafés sont de plus en plus nombreux à travers le pays. A chacun de voir si cet engouement est justifié. Pour notre part, nous nous attelons dans ce chapitre à vous présenter le monde de l'Internet comme un monde convivial, dans lequel les applications ne sont pas très compliquées à développer.

2. Le Web et l'Internet

2.1. La genèse

Le World Wide Web [LeWeb] a été créé en 1989 au CERN à Genève par Tim Berners-Lee. L'objectif premier du Web était de faciliter la communication au sein du CERN. En effet, les chercheurs sont répartis dans de nombreux pays et il était donc intéressant de développer un produit leur permettant un échange précis et rapide.

Le projet de départ était de réaliser un logiciel permettant la navigation hypertexte à travers un réseau. Il n'était nullement question de gérer des documents sonores, vidéo ou des images.

C'est seulement au début de l'année 1991 que la première démonstration de leur logiciel de navigation eut lieu à la conférence Hypertext '91. Depuis lors, l'engouement pour le Web

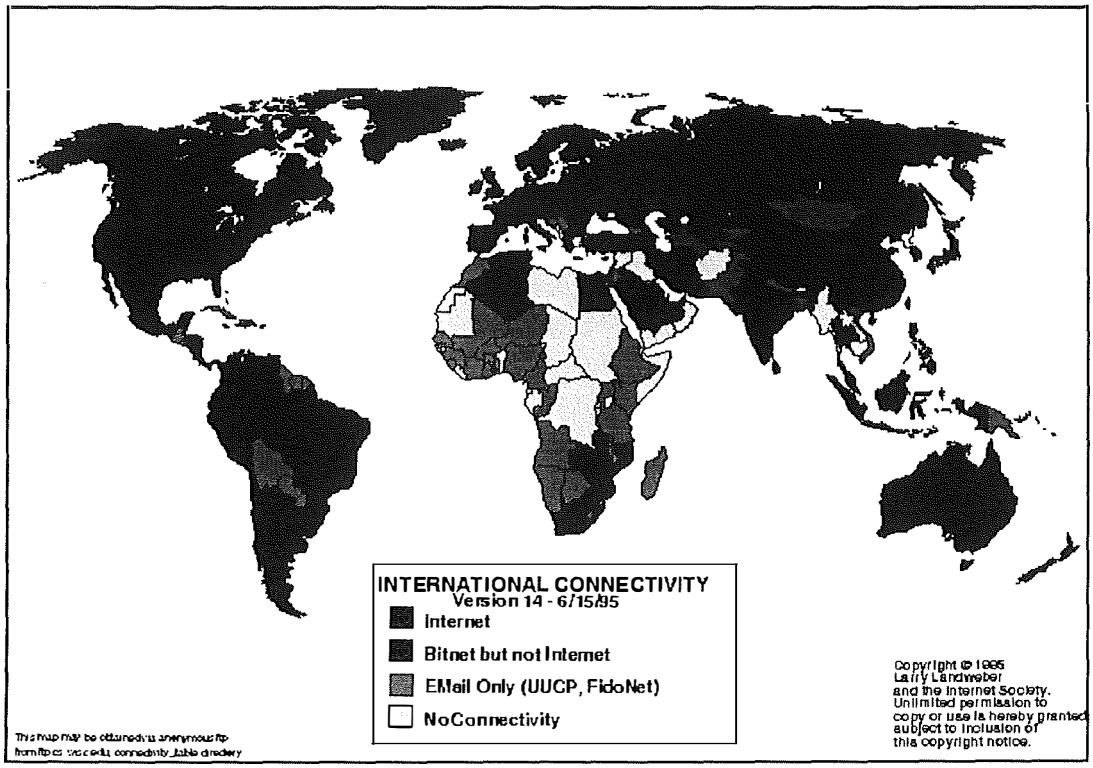


Figure 15 : Les connexions internationales

n'a cessé de croître. La *Figure 15* ci-dessous l'illustre à merveille.

Depuis l'idée créatrice, le Web a encore mûri. Le Web est maintenant vu comme une base de connaissances universelles. C'est un outil permettant la navigation hypermedia à travers des documents provenant du monde entier. Le projet du Web a donc fourni aux utilisateurs branchés sur le réseau Internet le moyen d'accéder à des informations variées d'une façon très simple.

2.2. L'hypertexte et l'hypermédia

Attardons-nous quelque peu sur les notions d'hypertexte et d'hypermédia. On les rencontre de plus en plus couramment.

Un ***hypertexte*** est fondamentalement de la même nature qu'un texte : on peut le visualiser, l'enregistrer et le modifier. Ils peuvent contenir tous les deux des références vers

d'autres textes. La grande différence est que dans le cas de l'hypertexte, **la liaison est en ligne**. Une simple sélection de la référence permet d'accéder au texte référence.

L'***hypermédia*** est proche de l'hypertexte en ce sens qu'il contient des liens vers d'autres documents. Par contre, les documents référencés dans un document hypermédia sont de types différents : **texte, son et image**. Par exemple, on peut ainsi lire une description des facultés de Namur qui contient un lien vers un discours du recteur, une photographie du bâtiment central ou un graphique représentant la population estudiantine de chaque faculté.

2.3. Le Web et l'Internet

Web et Internet sont deux mots qui sont souvent pris l'un pour l'autre et pourtant, ils ne représentent pas la même chose. L'Internet est un réseau d'une grandeur mondiale. Littéralement, ce mot signifie **un réseau de réseaux**. L'Internet rassemble des milliers de réseaux locaux éparpillés à travers le monde. C'est donc un ensemble de câbles et d'ordinateurs reliés entre eux.

Le Web, de son côté, représente **une collection de documents hypermédia**. Il ne faut donc pas confondre l'Internet et le Web. Celui-ci utilise l'Internet comme support de communication.

Dans les deux sections suivantes, nous allons présenter successivement le fonctionnement de l'Internet et le fonctionnement du Web.

2.4. Le fonctionnement de l'Internet

L'**architecture** de l'Internet est basée sur une collection de réseaux autonomes que l'on nomme les domaines de routage.

Le cœur de l'Internet est son **protocole** : TCP /IP. L'utilisation de TCP/IP permet une indépendance par rapport à la machine et au système d'exploitation. Ce sont des protocoles ouverts et distribués gratuitement. Enfin, ils sont basés sur un schéma d'adressage commun. La pile TCP/IP peut se subdiviser en couches du monde OSI, comme le montre la *Figure 16*.

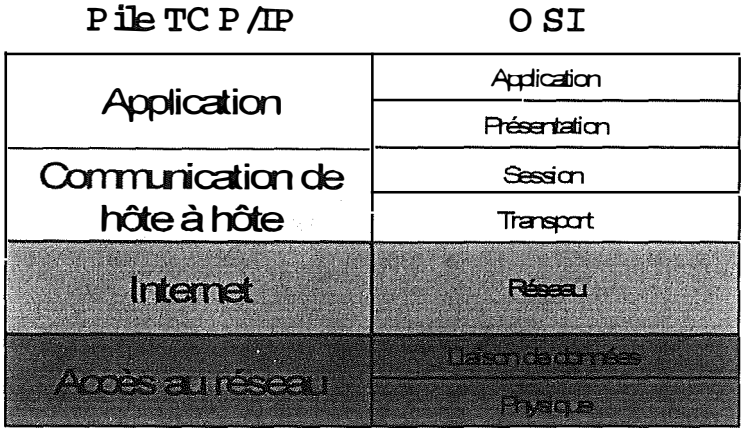


Figure 16 : La pile TCP/IP

La **couche d'accès au réseau** donne au système les moyens de délivrer les données aux machines connectées localement. Cette couche reçoit de la couche Internet des datagrammes et les transforme en trames qui seront transmises sur le réseau. La deuxième fonction principale de cette couche est la conversion des adresses Internet en adresses physiques du réseau local.

Le **protocole IP**, compris dans la couche Internet, fournit les services d'acheminement des paquets. Pour ce faire, il gère la définition des datagrammes, le routage des datagrammes jusqu'à leur destination, la définition du schéma d'adressage, et enfin, la fragmentation et le réassemblage des datagrammes.

IP est un protocole sans connexion; il garantit uniquement que les données ont été transmises au réseau. Les datagrammes IP ont une entête qui contient les adresses de source et de destination, des informations quant à la fragmentation éventuelle des données et des informations à propos du protocole utilisé.

Les adresses IP sont uniques. Elles sont sur 32 bits. Les bits de poids forts permettent d'identifier le réseau et les bits de poids faibles désignent des points d'accès (des machines). On retrouve trois classes d'adresses IP, comme nous l'illustrons dans la *Figure 17*, où les cases grises représentent la partie réseau et la blanche la partie machine.

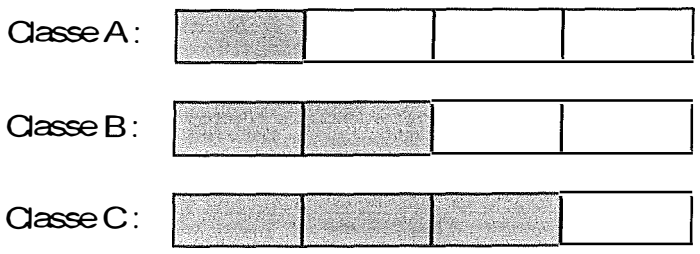


Figure 17 : Les classes d'adresses IP

La gestion des adresses Internet est hiérarchique : le NICC (Network Information Control Centre) distribue des fourchettes d'adresses à des organismes officiels qui se chargent d'allouer des adresses aux organismes demandeurs. La partie machine de l'adresse est gérée localement. Les trois types de formats d'adresse permettent une certaine souplesse vis-à-vis de la taille des réseaux physiques. Une adresse de classe A autorise 128 réseaux avec chacun 16 millions de points d'accès. Une adresse de classe B permet 16 mille réseaux et 64 mille machines. Enfin, la classe C assure 2 millions de réseaux et 256 mille machines.

Le principe du **routage** est relativement simple et est illustré à la *Figure 18*. Si l'adresse de destination du message que l'on envoie sur le réseau ne correspond pas à une adresse du réseau local, on envoie ce datagramme au routeur local qui se charge de faire passer l'information à un réseau local mitoyen.

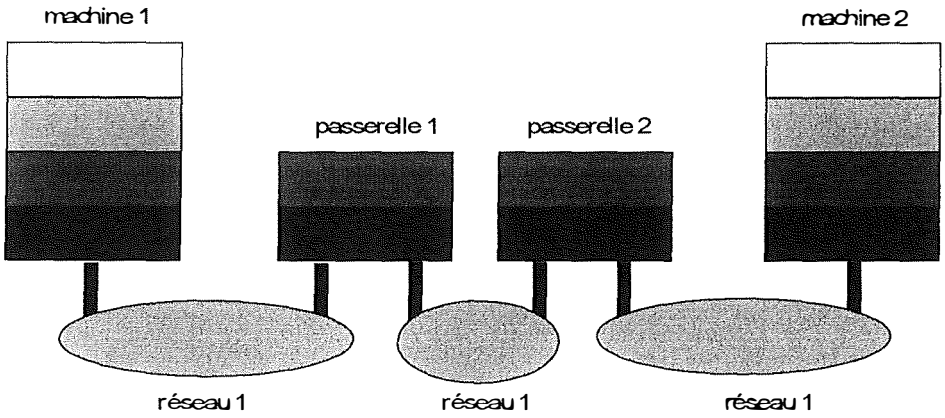


Figure 18 : Le routage

Pour **la couche de transport**, deux protocoles principaux existent : TCP et UDP. TCP propose un service de bout en bout, avec détection et correction des erreurs. UDP assure un service par datagrammes, sans connexion. Les développeurs peuvent choisir quel type de couche de transport convient le mieux à leur application.

Le **protocole TCP** utilise le mécanisme de Positive Acknowledgement Retransmission. Cela signifie que chaque segment émis est acquitté par le destinataire. Et si l'émetteur ne reçoit pas d'accusé de réception après un certain délai, il émettra le segment à nouveau. Les entêtes des segments comprennent : les ports sources et destination, le numéro de séquence et la taille de la fenêtre de réception.

2.5. Le fonctionnement du Web

Le Web est conçu à partir d'une **architecture client/serveur répartie**. Un Client Web, appelé browser, est un programme qui envoie des demandes de documents à des serveurs Web.

Un Serveur Web est un programme qui, lorsqu'il reçoit une requête d'un Client Web, envoie le document demandé ou un message d'erreur au Client. L'architecture est distribuée en ce sens que le programme du Client et du Serveur peuvent fonctionner sur des machines distantes. Le rôle du Serveur est de stocker les documents et celui du Client est de demander des documents et d'afficher les documents qu'il a reçu du Serveur. Le programme du Client et celui du Serveur peuvent donc fonctionner de façon autonome. En effet, le Serveur se contente d'attendre les demandes des Clinets. La charge de travail pour l'un comme pour l'autre n'est pas trop élevée. Car, le Serveur n'est actif que lorsqu'il reçoit une demande, le reste du temps, il est en attente. Le Client n'est forcément actif que lorsqu'il effectue des requêtes.

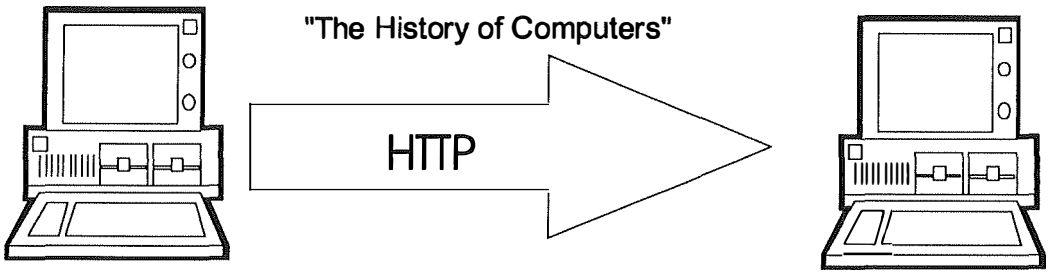
Voici un petit exemple présenté dans la **Erreur! Source du renvoi introuvable.** de dialogue établi entre le Client et le Serveur :

1. A partir d'un browser (le Client Web), un utilisateur sélectionne un hyperlien dans un document hypertexte ; il veut accéder au document : « The history of computers »;

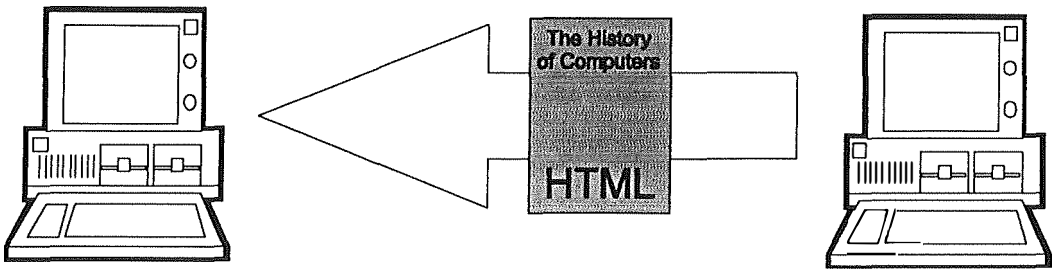
2. Le Client Web utilise l'adresse associée à l'hyperlien pour se connecter au Serveur Web. Il envoie une requête pour le document concerné au Serveur;

3. Le Serveur répond en fournissant au Client, le document en question. Il reste au Client à afficher le document.

Notions importantes :



Le client envoie un message HTTP, qui constitue une demande d'un document, à un serveur Web.



Le serveur envoie le document HTML au client. Celui-ci apparaît sur l'écran du client.

Figure 19 : Une transaction typique dans le Web

- Le langage de communication entre les clients et les serveurs Web est appelé **HyperText Transfer Protocol** (HTTP).

- L'architecture fonctionne en **mode non connecté** : il n'y a pas de session permanente entre le client et le serveur. Une transaction comprend de ce fait quatre étapes :
  1. la connexion du client sur le serveur;
  2. l'envoi d'une requête : un message du client au serveur;
  3. la réception d'une réponse du serveur;
  4. la fermeture de la connexion.

- Les documents sont référencés de façon unique grâce aux **Uniform Resource Locator** (URL) sur tout le réseau Internet [BLMM94]. A chaque hyperlien est associé un URL qui permettra au client d'établir la connexion avec le serveur et de spécifier à celui-ci quel document il désire consulter. La syntaxe d'un URL est standard :

  Une première partie fournit la méthode d'accès au document (file, http, ftp, telnet, ...).
  La seconde partie spécifie l'adresse IP de la machine (sous la forme d'une adresse IP ou d'un nom logique).
  La dernière partie comprend le chemin d'accès à la ressource.

Exemple :

http://hoohoo.ncsa.uiuc.edu/cgi/overview.html

http est la méthode d'accès. *hoohoo.ncsa.uiuc.edu* est le nom logique du serveur où se trouve le document. *cgi/overview.html* est le chemin d'accès à la ressource concernée.

3. Le langage HTML

Dans les paragraphes suivants, nous allons décrire les éléments de base de l'HyperText Markup Language (HTML) [BLC95] [BLFF95]. Ce langage est utilisé pour **la rédaction des documents Web**. Il est basé sur l'utilisation d'étiquettes de formatage. La facilité d'utilisation de ce langage est un des facteurs qui concourt au succès croissant du Web.

HTML et surtout HTML+ permettent la réalisation d'interfaces conviviales. Il existe de nombreux produits sur le marché qui facilitent la création de documents Web. Des programmes effectuent immédiatement la conversion entre de nombreux formats (Word, Framemaker, Latex, ...) et le langage HTML. Enfin, HTML offre la possibilité de créer des formulaires. Nous étudierons plus en détails la gestion des formulaires dans la section suivante.

3.1. Le schéma général d'un document

```
<HEAD>
<TITLE> Le titre du document </TITLE>
</HEAD>
<BODY>
<H1> Le format des titres </H1>
<H2> La mise en forme élémentaire </H2>
<P> Ceci définit un nouveau paragraphe. </P>

<P> Le second paragraphe présente les différents types de
caractères qui sont définis, on retrouve des textes en <I>
italique</I>et d'autres en <B > gras </B>. <P>

<P> Il est possible de créer des listes :< /P>
<UL>
<LI> le premier élément de la liste
<LI> le second
</UL>

<H2> Les ancres et les images </H2>

<P>Pour visiter RD2P <A
href= "http://puligny:10000/rd2p.html">cliquer ici </A></P>

<P>Nous pouvons insérer ici le logo de RD2P : <IMG
SRC="rd2p.gif"></P>
</BODY>
```

La *Figure 20* nous présente le résultat du document tel que nous l'avons défini. Nous constatons que tout document se découpe en une entête, comprenant le titre du document, et un corps. Nous allons étudier dans les paragraphes suivants les éléments de mise en forme plus en détail.

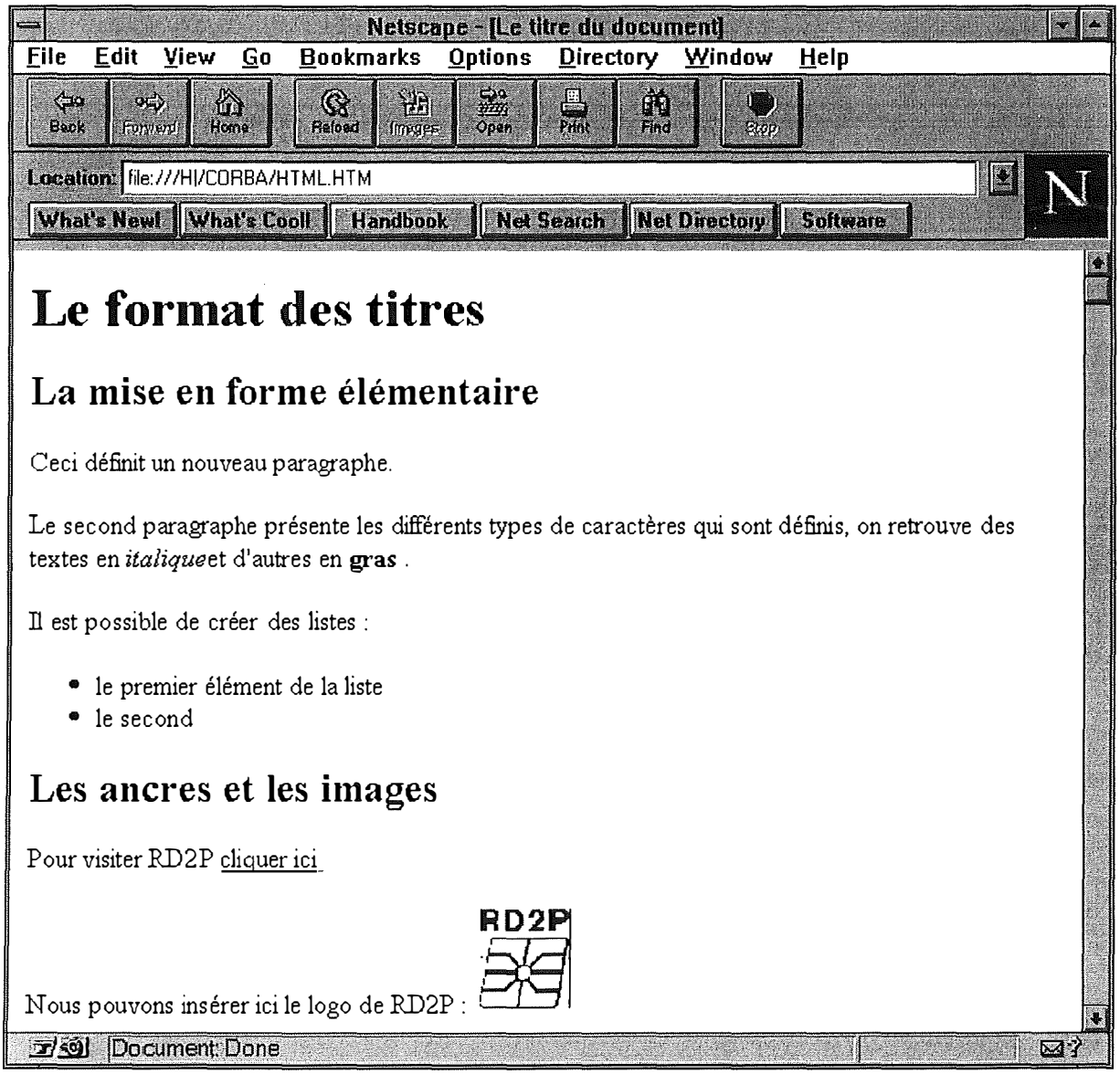


Figure 20 : Un exemple de page HTML

3.2. La mise en forme de documents HTML

a) Les titres, les paragraphes et les tabulations

L'étiquette <Hn> permet de définir un titre de niveau n. Il existe six niveaux de titres différents <H1> ... <H6>. Chaque titre se termine par l'étiquette </Hn>

Un paragraphe est défini par les étiquettes <P> et </P>. Il est toutefois possible de ne pas placer d'étiquette de début.

Les tabulations sont placées par l'étiquette <BLOCKQUOTE> et sont supprimées par </BLOCKQUOTE>.

b) Les caractères

Pour changer la forme des caractères, deux possibilités s'offrent aux créateurs de documents. Soit ils utilisent des types de caractères prédéfinis que l'on appelle les caractères logiques, soit ils utilisent leurs propres styles; on parle alors de **caractères physiques**.

Les caractères logiques sont au nombre de sept. On les utilise de façon standard c'est-à-dire avec une étiquette au début et une étiquette finale. Voici les utilisations proposées; libre à chacun de moduler ces utilisations suivant ses goûts :

- <DFN> et </DFN> sont utilisés pour les définitions
- <CITE> et </CITE> pour les citations
- <CODE> et </CODE> pour du code
- <KBD> et </KBD> pour les entrées au clavier
- <SAMP> et </SAMP> pour les messages de l'utilisateur
- <STRONG>,</STRONG>, <EM> et </EM> pour faire ressortir un texte

Pour les caractères physiques, on a la possibilité de mettre un texte en gras, en italique, à une certaine taille et enfin de lui laisser sa forme initiale. Pour cela, on utilise respectivement les étiquettes <B>, <I>, <SIZE> et <PRE>.

c) Les listes

Le langage HTML permet la création de **liste numérotée** et de **liste avec des puces**. Les éléments des listes se placent derrière l'étiquette de début unique <LI>. Les étiquettes <UL> et <OL> entourent respectivement les éléments des listes à puces et numérotées.

Exemple :

```
<UL>
<LI> premier élément de la liste
<LI> le second élément
</UL>
```

d) Les ancres et les images

Une ancre est un moyen d'accès à une autre ressource, quel qu'en soit le type. On peut définir des ancres à l'intérieur de son document. Pour les documents volumineux, il est ,par exemple, intéressant de pouvoir indexer les chapitres. Pour les ancres internes, il faut placer des références dans le document de la forme <A NAME="**nom_référence**"> Chapitre 1</A>. Ensuite, on peut poser des ancres qui ont la forme suivante : <A HREF="#**nom_référence**"> accès au chapitre 1</A>.

Les ancres vers des documents externes sont composés de la manière suivante : <A HREF=le chemin d'accès au document référencé> le nom de l'ancre </A>.

Les images sont soit incluses immédiatement dans le document soit référencées. Dans le premier cas, on peut décider de l'alignement de l'image : en-dessous, au-dessus ou au milieu de son point d'insertion. La syntaxe est très simple : <IMG SRC= le chemin d'accès à l'image>. Par défaut l'alignement est en-dessous. Si l'on désire le changer, on introduit dans l'étiquette, juste après IMG, les mots ALIGN=TOP ou MIDDLE afin de positionner l'alignement respectivement au-dessus et au milieu.

Dans le cas où l'image est référencée, il s'agit uniquement d'indiquer dans l'ancre le chemin d'accès à l'image.

Notons enfin qu'il est possible pour une image de jouer le rôle d'une ancre. Il suffit de structurer son texte de la façon suivante : <A HREF=le chemin d'accès au document référencé> <IMG SRC= le chemin d'accès à l'image> </A>.

e) Divers

Les étiquettes <BR> et <HR> permettent respectivement d'insérer un espace horizontal et une barre horizontale afin de structurer davantage le document.

3.3. Les formulaires

Les formulaires sont très utiles car ils permettent **le dialogue entre le client et le serveur grâce aux CGI**, comme nous le verrons dans le point suivant. Il y a deux types de méthode pour gérer les formulaires : la méthode POST et la méthode GET. La méthode POST insère les résultats du formulaire dans une variable globale qui est communiquée au serveur. De son côté, la méthode GET insère immédiatement les résultats dans l'URL. Il est toujours préférable d'utiliser la méthode POST car elle n'est pas limitative. Par contre, la méthode GET n'autorise l'insertion que de 256 caractères.

Voici le schéma général :

<FORM METHOD=POST/PRE ACTION="le chemin d'accès au CGI">
.
.
</FORM>

a) Les champs d'édition

Les champs d'édition unilinéaire et multilinéaire sont acceptés par HTML. Dans tous les paragraphes suivants, un mot en *italique* est un attribut facultatif. Voici un petit exemple illustrant une utilisation possible des champs d'édition :

```
<P>Nom : <INPUT TYPE=TEXT NAME="nom" VALUE="Dupont" SIZE=20
MAXLENGTH=30></P>
<P>Mot de passe : <INPUT TYPE=PASSWORD NAME="mot-passe" SIZE=8
MAXLENGTH=8></P>
<P>Adresse:<TEXTAREA NAME="adr" COLS=64 ROWS=5></P>
```

L'argument VALUE permet d'insérer un texte par défaut. La taille du champ d'édition est spécifié par l'argument SIZE. Le champ MAXLENGTH correspond au nombre maximum de caractères que l'on peut entrer dans le champ d'édition. Lorsque l'on insère des caractères dans un champ de type PASSWORD, ceux-ci sont cachés par des étoiles ('*'). TEXTAREA est l'étiquette utilisée pour les champs d'édition multilinéaire. COLS définit le nombre de colonnes et ROWS le nombre de lignes.

b) Les boîtes à cocher et les boutons radio

Les boîtes à cocher présentent un certain nombre d'options parmi lesquelles on peut faire un choix booléen. Par contre, dans les boutons radio, on ne peut choisir qu'une seule option parmi celles qui sont proposées.

```
<INPUT TYPE=CHECKBOX NAME="boite1" VALUE="ok" CHECKED>classe A
<INPUT TYPE=CHECKBOX NAME="boite2" VALUE="oui" >classe B

<INPUT TYPE=RADIO NAME="radio1" VALUE="sur" CHECKED> avec certitude
<INPUT TYPE=RADIO NAME=" radio2" VALUE="doute"> peut-être
```

Le nom et la valeur sont utilisés par les CGI pour gérer le résultat. CHECKED indique que cette option est sélectionnée par défaut.

c) Les boîtes de sélection

Les boîtes de sélection permettent de choisir un ou plusieurs éléments parmi une liste déterminée à l'avance.

```
<SELECT NAME="ville" SIZE=3 MULTIPLE> Ville d'origine :
<OPTION SELECTED> Namur
<OPTION> Charleroi
<OPTION> Lille
<OPTION> Bruxelles
</SELECT>
```

La taille représente le nombre d'éléments qui seront affichés dans la boîte de sélection. L'attribut facultatif MULTIPLE signifie que l'utilisateur peut sélectionner plusieurs éléments de la liste. Enfin, si l'attribut SELECTED est présent dans l'étiquette OPTION, la valeur qui suit sera sélectionnée par défaut.

d) Les boutons de commandes

Il existe deux types de boutons de commande; les boutons de soumission et les boutons de remise à l'état initial. Si le bouton de remise à zéro est sélectionné, les champs sont mis à leur valeur par défaut. Le bouton de sélection valide les données insérées et les envoie au CGI concerné.

<INPUT TYPE=**SUBMIT** NAME="bouton ok" VALUE="OK">
<INPUT TYPE=**RESET** NAME="bouton reset" VALUE="Recommencer">

La valeur correspond à l'intitulé du bouton de commande.

4. Les Common Gateway Interface

Un Common Gateway Interface (CGI) est **un standard pour interfacer des informations entre un serveur Web et un programme** [McC95]. Les CGI permettent de rendre les documents HTML **plus dynamiques**. En effet, un document HTML est toujours composé d'un texte figé. Par contre les CGI sont exécutés en temps réel et ils peuvent de ce fait créer dynamiquement[4] des documents à partir des informations reçues.

Par exemple, il est de plus en plus souvent question de s'inscrire avant de pouvoir accéder à un site Web. Dans ce cas, on reçoit un formulaire à remplir. Suivant les réponses que l'on a fournies, le CGI acceptera l'inscription, la refusera ou bien lui demandera des informations complémentaires.

Les CGI peuvent être écrits en plusieurs langages : C, C++, Fortran, PERL, Unix shell, Visual Basic, AppleScript. Les programmes ainsi écrits sont placés dans le répertoire /cgi-bin de son browser. Le plus souvent ou utilisera un langage interprété quand il s'agit d'un script qui doit être souvent modifié.

[4] Cfr. [MGG96].

4.1. Le lien entre les clients, les CGI et le serveur

Nous allons expliquer dans les paragraphes suivants le fonctionnement des CGI en tant qu'interface entre le client et le serveur. La *Figure 21* illustre cela parfaitement.

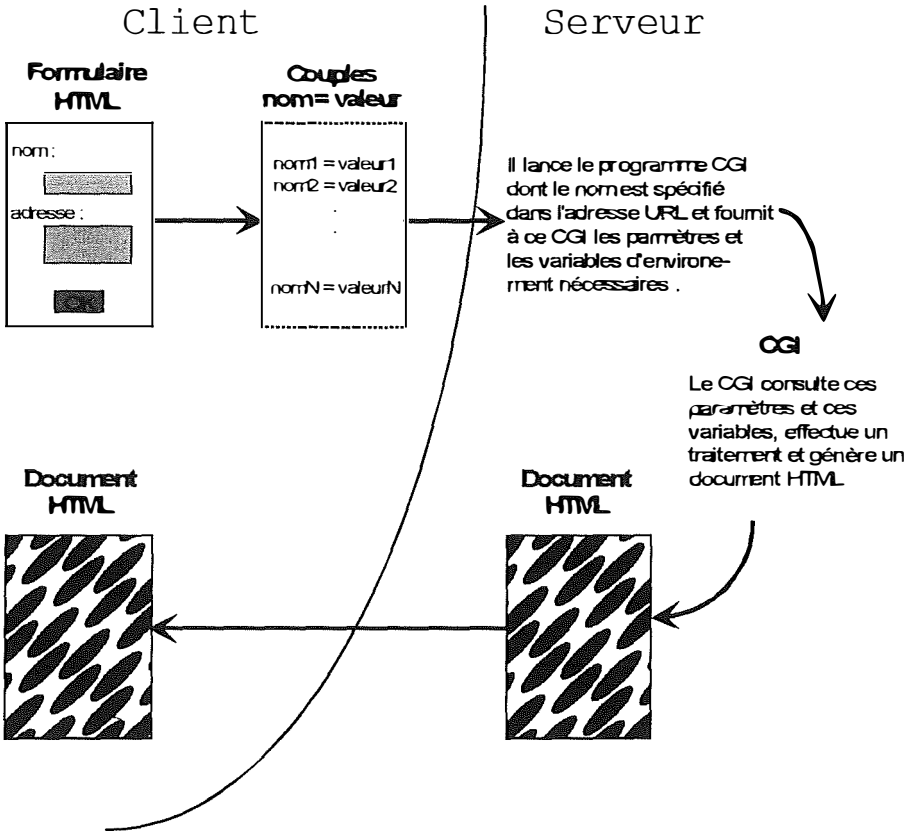


Figure 21 : Les liens entre le client, le serveur et le CGI

Le client a devant lui un formulaire HTML qu'il remplit. Les données, ainsi entrées dans le formulaire, sont assemblées par le browser Web en **une série de paires nom/valeur**. Ces paires sont envoyées au serveur pour être traitées par le CGI spécifié dans l'attribut ACTION de l'étiquette FORM. Comme nous l'avons déjà expliqué, soit directement dans l'URL (dans le cas d'une méthode GET) soit dans une variable (pour les méthodes POST).

Le serveur lance le programme CGI qui a comme paramètres en entrée les couples nom/valeur. Le programme CGI consulte à la fois les paramètres, qui lui ont été envoyés sur son entrée standard, et les variables globales. Il peut alors effectuer toute une série de tâches telles que la mise à jour ou la consultation d'une base de données, la lecture d'une carte à microprocesseur, l'envoi d'un mail... Toutes ces opérations sont exécutées sur le serveur. En fin de compte, le CGI génère un document HTML qu'il envoie sur sa sortie standard. Le client reçoit et affiche la réponse.

Le traitement des paires nom/valeur par le CGI est facilité par des bibliothèques de fonctions qui existent dans tous les langages. Par exemple, le message reçut par le CGI (dans le cas de méthodes GET), est de la forme suivante : *nom_cgi?nom=SCIEUR&adresse=rue+des+roses*. Ce message est facilement décodable.

4.2. Les variables d'environnement

Dans les paragraphes suivants, nous allons décrire les variables d'environnement. Celles-ci servent, tout comme les paramètres, à gérer les interactions entre un programme CGI et un serveur Web. Il existe tout d'abord trois variables qui sont utilisées quel que soit le type de requête :

- `SERVER_SOFTWARE` : le nom et la version du serveur qui traite les requêtes.
- `SERVER_NAME` : le nom de la machine sur laquelle s'exécute le serveur. C'est l'identifiant utilisé dans la première partie de l'URL soit une adresse IP soit un alias (info.fundp.ac.be).
- `GATEWAY_INTERFACE` : la version des CGI exécutables sur le serveur.

Suivant les cas, d'autres variables d'environnement interviennent également; nous décrivons les principales :

- `SERVER_PROTOCOL` : le nom et la version du protocole utilisé par l'ordinateur qui a effectué la requête (exemple : HTTP/1.0).
- `SERVER_PORT` : le numéro de port sur lequel la demande a été reçue.
- `REQUEST_METHOD` : la méthode avec laquelle la requête a été réalisée (exemple : POST, GET)
- `QUERY_STRING` : les informations qui suivent le point d'interrogation dans l'URL. Cette variable est utilisée par la méthode GET qui, comme nous l'avons déjà expliqué, place les paires nom/valeur dans l'URL à la suite du nom du CGI.
- `REMOTE_HOST` : le nom de l'ordinateur hôte (celui qui fait la requête).
- `REMOTE_ADDR` : l'adresse IP de ce même hôte.
- `CONTENT_TYPE` : les données d'un formulaire de type POST sont placées, toujours sous la forme de paires, dans cette variable.
- `CONTENT_LENGTH` : la longueur de la variable `CONTENT_TYPE`.

5. Conclusions

Le monde de l'Internet nous apparaît comme très riche. Il permet de manipuler très facilement des ressources de plus en plus complexes. Les CGI, notamment, assurent une grande flexibilité et garantissent un caractère dynamique aux applications qui sont développées. Tout au long de la mise en place de notre application, la syntaxe HTML nous a semblé très aisée à manipuler. Toutefois, le langage HTML+ apporte des réponses à certains manquements qui nous sont apparus au cours de nos manipulations. L'Internet risque de connaître un développement supplémentaire grâce au langage JAVA très prometteur.

Chapitre IV :
Le développement du logiciel

1. Introduction

Dans les chapitres précédents, nous nous sommes attachés à expliciter d'une façon concise les différents domaines touchés par notre application. Tout au long de ce chapitre, nous présentons d'une manière précise le logiciel que nous avons développé.

La première section précise la méthode que nous avons utilisée pour le développement de ce logiciel. Ensuite, nous décrivons les spécifications, qui découlent du cahier des charges. Celles-ci comprennent l'élaboration d'un schéma entité-association, d'un schéma fonctionnel et enfin de la structure des tables. La conception physique de l'application est le sujet de la section suivante. Nous y détaillerons les droits d'accès aux différentes tables, l'enchaînement des CGI et l'interface homme-machine. Enfin, la partie implémentation est l'objet de la dernière section. On y parle de la plate-forme Orbix. On y présente un modèle de CGI et des exemples de fenêtres de l'application.

2. La méthode

Nous avons utilisé **une approche transformationnelle par prototypage**. Ce type d'approche est illustrée par la *Figure 22.*

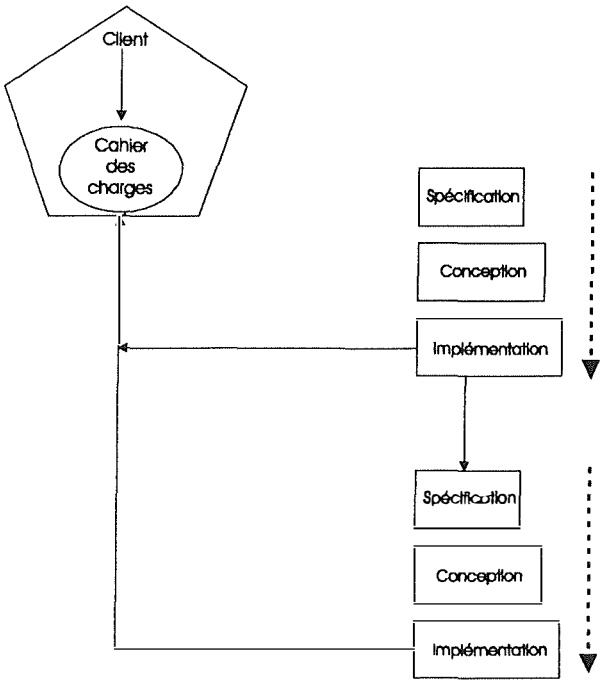


Figure 22 : L'approche par prototypage

Un certain nombre de cycles transformationnels ont eu lieu, comprenant les étapes suivantes : spécification, conception et implémentation. Ces cycles se déroulent en complète concertation avec le cahier des charges. Chaque cycle donne lieu à un prototype qui est soumis au client. Le client émet une critique vis-à-vis du prototype. En fonction de cette critique, on peut améliorer le logiciel pour qu'il corresponde mieux aux desiderata du client. Lors de notre projet, on peut estimer que trois ou quatre évaluations ont été appliquées.

3. La spécification

3.1. Modèle entité-association

Cette partie a pour but de décrire le modèle entité-association que nous avons déduit du cahier des charges. Pour construire ce modèle, nous nous sommes inspirés d'un travail précédemment réalisé [Sab93] à propos d'OpenCard en milieu hospitalier.

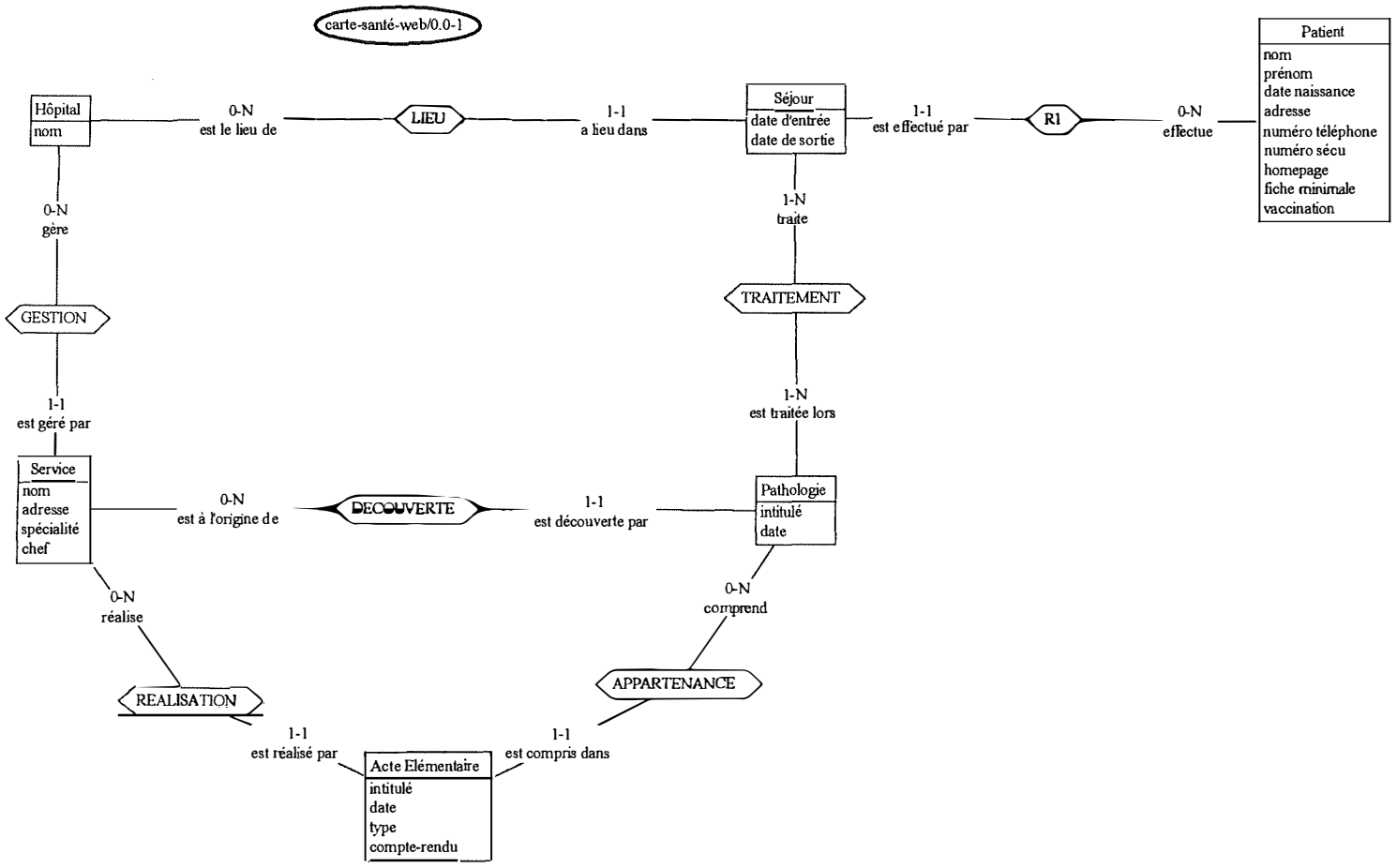


Figure 23 : Le schema entité-association

Nous avons dégagé, comme nous le montrons dans la *Figure* 23, six types d'entités : HOPITAL, SEJOUR, PATIENT, PATHOLOGIE, ACTE ELEMENTAIRE et SERVICE.

Les relations entre ces entités sont les suivantes :

- Un HOPITAL *est le lieu de* 0-N SEJOUR.
- Un SEJOUR *a lieu dans* 1-1 HOPITAL.
- Un PATIENT *effectue* de 0-N SEJOUR.
- Un SEJOUR *est effectué par* 1-1 PATIENT.
- Un SEJOUR *traite* 1-N PATHOLOGIE.
- Une PATHOLOGIE *est traitée par* 1-N SEJOUR.

- Une PATHOLOGIE *entraîne* de 0-N ACTE ELEMENTAIRE.
- Un ACTE ELEMENTAIRE *est entraîné par* 1-1 PATHOLOGIE.
- Un ACTE ELEMENTAIRE *est opéré dans* 1-1 SERVICE.
- Un SERVICE *opère* 0-N ACTE ELEMENTAIRE.
- Un SERVICE *est à l'origine de* 0-N PATHOLOGIE.
- Une PATHOLOGIE *est découverte par* 1-1 SERVICE.
- Un SERVICE *est géré par* 1-1 HOPITAL.
- Un HOPITAL *gère* de 0-N SERVICE.

3.2. Les types d'attribut

Nous reprenons chaque type d'entité afin d'en spécifier le contenu. La première colonne contient l'intitulé et la deuxième la longueur. Notons que cette longueur représente le nombre caractères. En effet, comme nous l'avons vu précédemment (au point 3.3.1), seul le type '*caractère*' est connu de la carte CQL.

Le type d'entité HOPITAL :

NOM	20

Le type d'entité SEJOUR :

DATE D'ENTREE	8
DATE DE SORTIE	8

Le type d'entité PATIENT :

NOM	20
PRENOM	20
ADRESSE	20
DATE DE NAISSANCE	8
NUMERO DE TELEPHONE	12
NUMERO DE SECURITE SOCIALE	12
HOME PAGE	20

Le type d'entité PATHOLOGIE :

INTITULE	20
DATE DE LA DECOUVERTE	8

Le type d'entité ACTE ELEMENTAIRE :

INTITULE	20
DATE	8
TYPE	20
COMPTE-RENDU	20

Le type d'entité SERVICE :

NOM	20
ADRESSE	20
SPECIALITE	20
NOM DU CHEF	20

3.3. Le schéma fonctionnel

De ce modèle entité-association, nous déduisons le schéma fonctionnel (cfr. la *Figure 24*). Pour ce faire, nous avons ajouté une entité PASEJ qui permet d'éliminer la relation *N-N* qui existait entre les entités PATHOLOGIE et SEJOUR. Nous disposons maintenant d'un schéma composé uniquement de relations fonctionnelles.

Pour des raisons de clarté et de temps d'accès, nous scindons l'entité PATIENT en quatre entités : les INFORMATIONS ADMINISTRATIVES, la FICHE MINIMALE, les ALLERGIES et les VACCINATIONS.

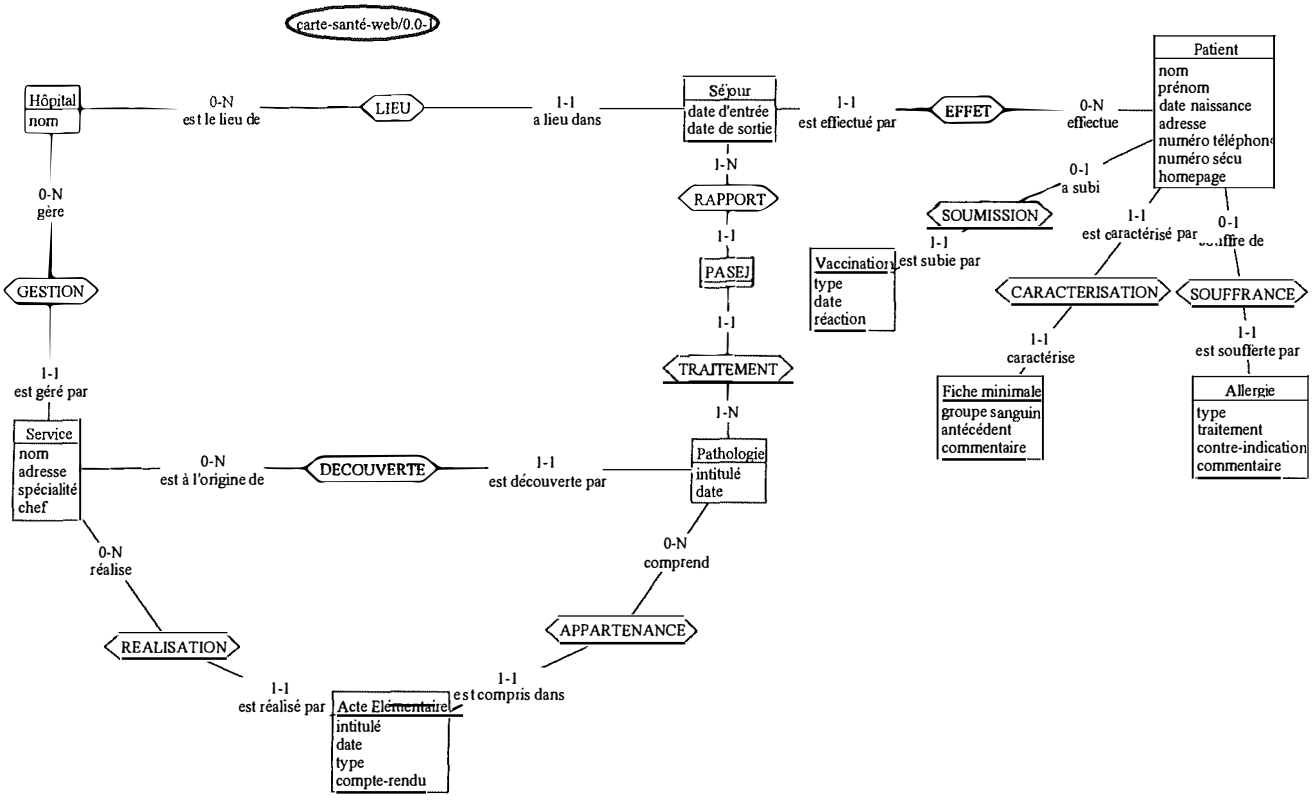
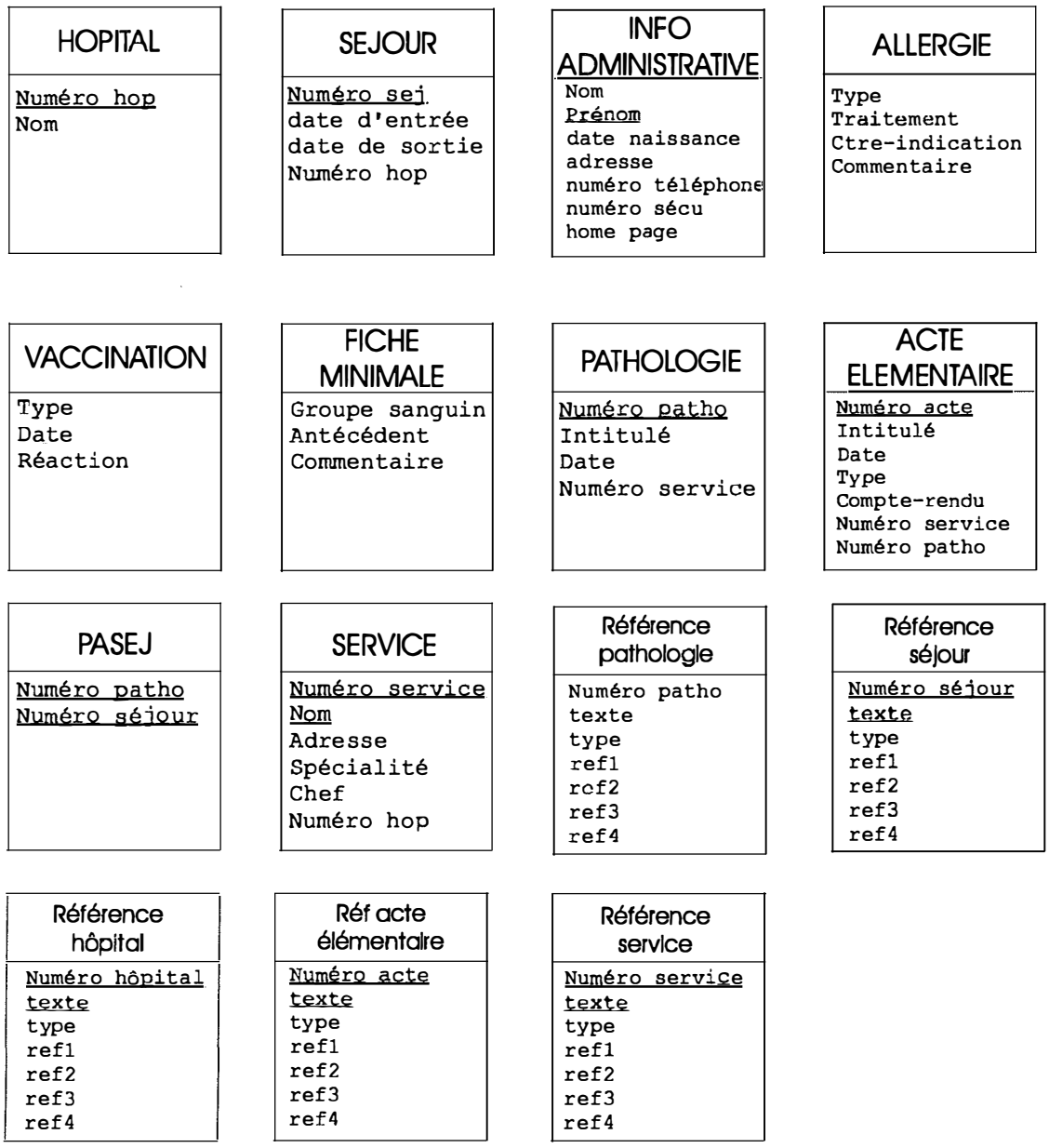


Figure 24 : Le schema fonctionnel

3.4. Les tables

A partir du schéma fonctionnel de la *Figure 24*, nous pouvons déduire la structure des tables qui composera notre système d'information.La correspondance entre les types d'entités et les tables est immédiate. On retrouve donc dix tables de base dans la *Figure 25*. A ces tables, **nous avons ajouté des tables de références** pour les informations médicales les plus importantes : à savoir l'hôpital, la pathologie, l'acte élémentaire, le service et le séjour.



CONTRAINTES REFERENTIELLES

tout Numéro hop **de** SEJOUR **est un** Numéro hop **d'**HOPITAL
tout Numéro service **de** PATHOLOGIE **est un** Numéro service **de** SERVICE
tout Numéro service **de** ACTE ELEMENTAIRE **est un** Numéro service **de** SERVICE
tout Numéro patho **de** ACTE ELEMENTAIRE **est un** Numéro patho **de** PATHOLOGIE
tout Numéro patho **de** PASEJ **est un** Numéro patho **de** PATHOLOGIE
tout Numéro séjour **de** PASEJ **est un** Numéro sej **de** SEJOUR
tout Numéro hop **de** SERVICE **est un** Numéro hop d'HOPITAL
tout Numéro hôpital **de** REFERENCE HOPITAL **est un** Numéro hop **d'**HOPITAL
tout Numéro acte **de** REFERENCE ACTE ELEMENTAIRE **est un** Numéro acte **d'**ACTE ELEMENTAIRE
tout Numéro patho **de** REFERENCE PATHOLOGIE **est un** Numéro patho **de** PATHOLOGIE
tout Numéro séjour **de** REFERENCE SEJOUR **est un** Numéro sej **de** SEJOUR
tout Numéro service **de** REFERENCE SERVICE **est un** Numéro service **de** SERVICE

Figure 25 : La structure des tables

Ces tables de références ont toutes la même structure. Elles ont pour objectif de contenir les différentes adresses URL auxquelles sont situés les objets référencés. Quatre colonnes sont réservées pour stocker une adresse. En effet, la taille d'une URL peut être importante.

4. Conception physique

4.1. Les droits d'accès

Suite à une consultation des professionnels de la santé, nous avons établi **une matrice d'accès aux informations**. Le *Tableau 6* reprend cette matrice. Les types d'accès possibles sont les suivants : sélection (S), insertion (I), mise-à-jour (M) et suppression (D). Tous représentent l'ensemble des droits d'accès.

Nom de la table :	Médecin	Infirmière	Hôtesse
Informations administratives	S	S	Tous
Acte élémentaire	Tous	-	-
Allergie	Tous	S	S,I
Fiche minimale	Tous	S	S,I
Hôpital	S	S	Tous
Pathologie-Séjour	Tous	-	-
Pathologie	Tous	-	-
Référence acte élémentaire	Tous	-	-
Référence hôpital	S	S	Tous
Référence pathologie	Tous	-	-
Référence séjour	Tous	-	Tous
Référence service	S,I	-	Tous
Séjour	Tous	-	Tous
Service	Tous	-	Tous
Vaccination	Tous	S	S,I

Tableau 6: La matrice des accès

4.2. L'enchaînement des CGI

Dans les figures ci-dessous, nous avons illustré l'enchaînement des principaux CGI. Dans un souci de clarté, nous nous sommes limités aux CGI concernant la consultation des données. Au total, ce sont trente-neuf CGI qui ont été développés pour notre application.

D'une façon globale, les CGI regroupent deux aspects : l'accès aux données sur la carte CQL et l'interfaçage de ces données à travers un browser Web.

Un membre du personnel médical prend possession de la carte du patient. Il l'insère dans le lecteur. Il a devant lui son browser avec la page Web-Santé. Il se connecte à la carte en sélectionnant le lien « connexion ».

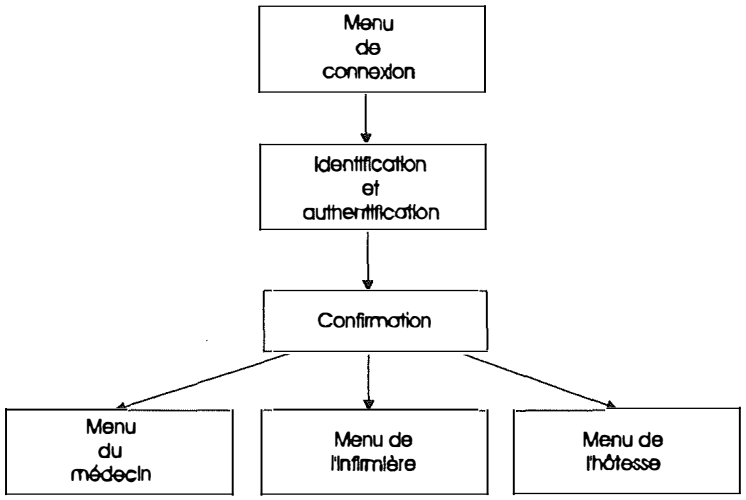


Figure 26 : La connexion

Il s'identifie comme un médecin, une infirmière ou une hôtesse. Il s'authentifie par son mot de passe. En appuyant sur le bouton « OK », les informations sont envoyées au CGI qui effectue la vérification de la présentation.

Si la présentation est erronée, un message d'erreur est affiché. Dans le cas contraire, un menu contextuel est présenté, suivant qu'il s'agisse d'un médecin, d'une infirmière ou d'une hôtesse.

Le menu du médecin lui offre quatre possibilités. Premièrement, il peut consulter **les informations administratives** concernant le patient. Deuxièmement, il a accès à **une fiche minimale du patient**. Celle-ci reprend des données médicales telles que les vaccinations, les allergies, le groupe sanguin, ... Ensuite, le médecin peut désirer toutes les informations au sujet **d'une pathologie particulière**. Enfin, une liste de **toutes les pathologies**, dont le patient a souffert jusqu'à présent, est accessible au médecin.

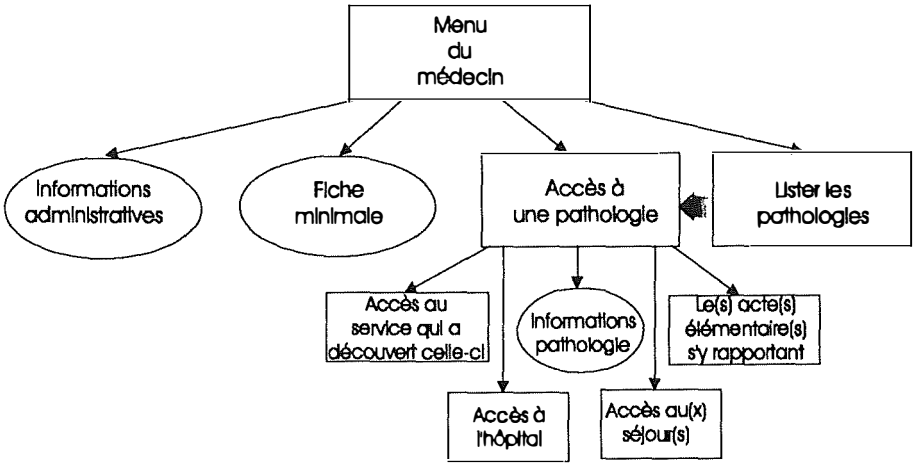


Figure 27 : Le menu du médecin

Les liens hypertextes nous permettent de voyager d'une table à l'autre de la carte sans que l'utilisateur ne s'en rende compte. Ainsi, les deux derniers choix du menu médecin permettront un accès à diverses informations ; le(s) acte(s) élémentaire(s) entrepris pour la pathologie concernée, le service qui a découvert celle-ci, l'hôpital dans lequel se trouve ce service et le(s) séjour(s) durant le(s)quel(s) le patient a été traité pour cette pathologie.

Comme nous l'avons expliqué précédemment, à chaque table importante est associée une table des références. Au fur et à mesure de nos pérégrinations à travers les CGI, **les références sont ajoutées aux informations de base**. Ainsi, un CGI qui consulte les actes élémentaires**, affichera également un lien vers les références à ces actes**. Une visite chez un pneumologue pourrait par exemple conduire celui-ci à effectuer un certain nombre de radiographies. Ils pourraient donc placer ces radiographies en référence à sa consultation.

Le médecin a le droit de modifier et d'ajouter toutes les informations médicales qu'il désire. Nous n'avons pas présenté l'enchaînement des CGI concernant ces mises-à-jour afin de ne pas alourdir la présentation. En pratique, à chaque fenêtre de consultation sont associées deux ou trois fenêtres de mises-à-jour.

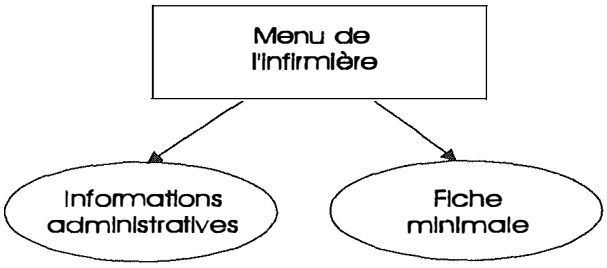


Figure 28 : Le menu de l'infirmière

Pour sa part, l'infirmière peut consulter **les informations administratives et la fiche minimale du patient**. Elle ne possède aucun droit de mise-à-jour ni d'insertion.

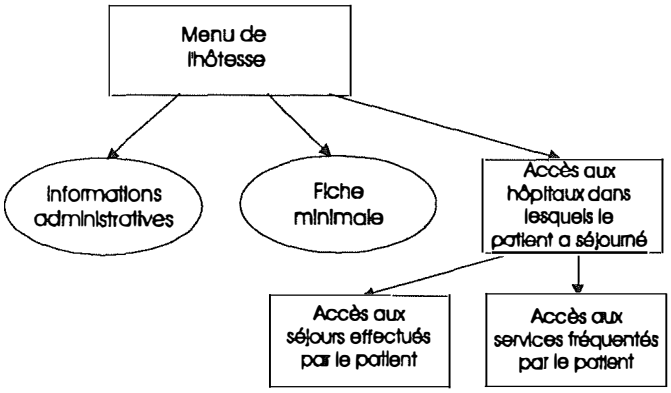


Figure 29 : Le menu de l'hôtesse

L'hôtesse a un rôle plutôt administratif. C'est elle qui accueille les patients à leur arrivée dans l'hôpital. Elle rédige donc la première **fiche minimale et insère les informations administratives** concernant le patient. Elle insère les données à propos du séjour du patient. Elle peut également ajouter **un service** à l'hôpital et ajouter des références **aux hôpitaux** et aux services.

4.3. Interface homme-machine

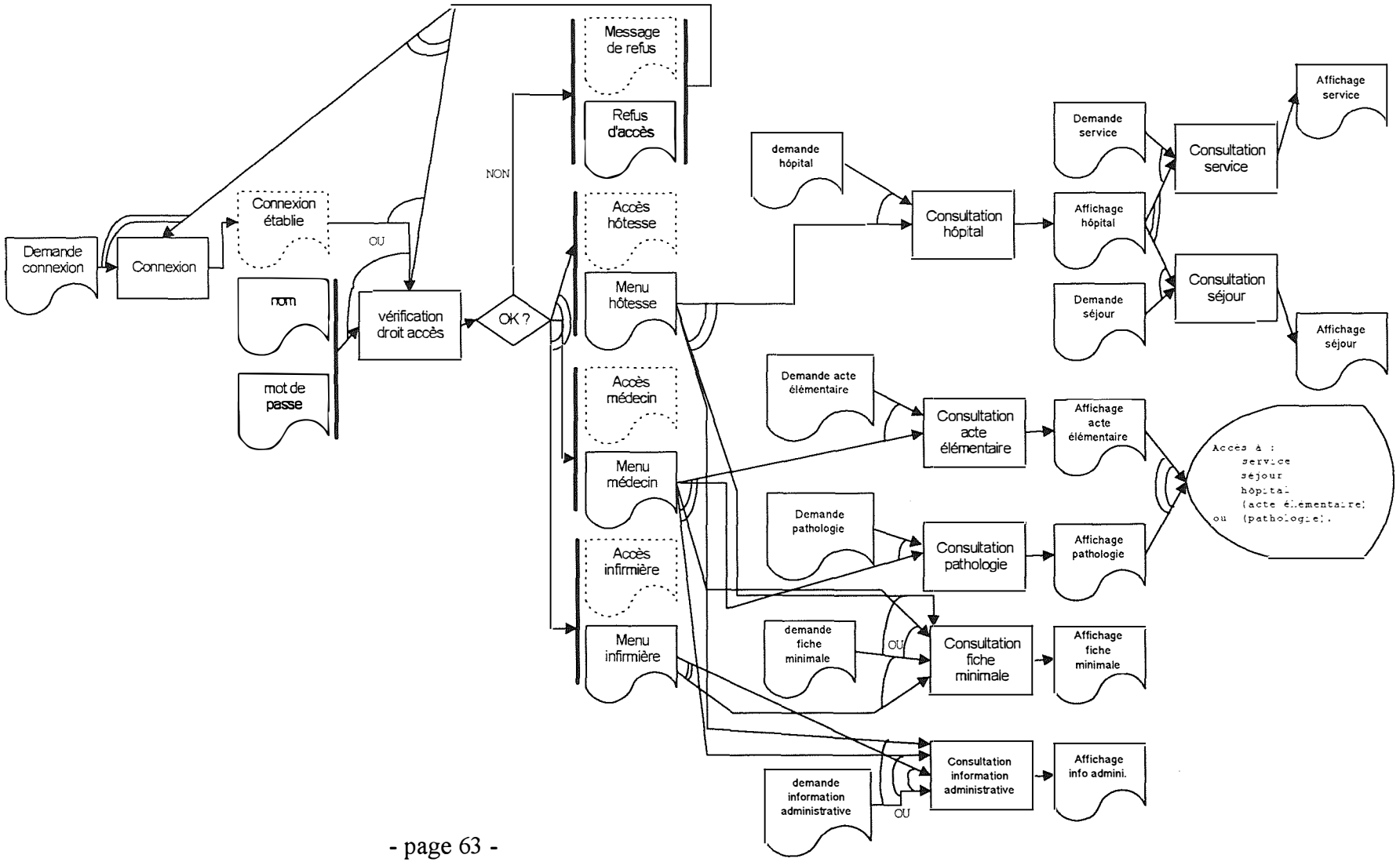


Figure 30: Le graphe d'enchaînement des fonctions

La *Figure 30* de la page précédente présente le graphe d'enchaînement des fonctions. Nous avons réalisé celui-ci afin de mettre en évidence les unités de présentation dans un premier temps et les fenêtres dans un second. Pour ce faire, nous nous sommes basés sur la méthode proposée par l'équipe Trident [BHLPVZ94].

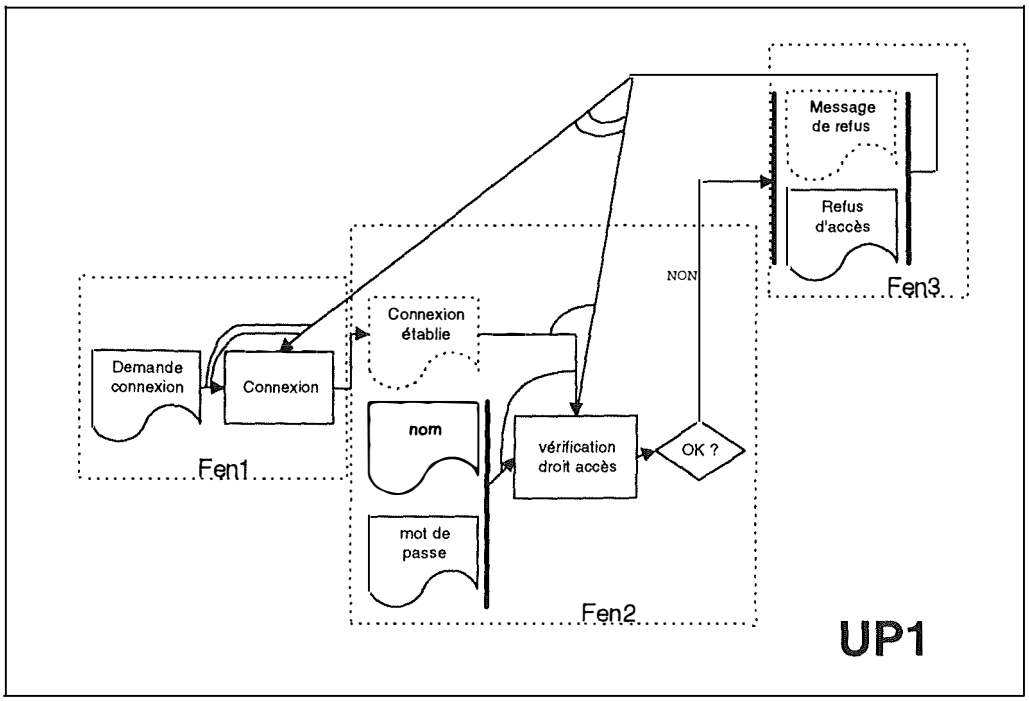


Figure 31 : L'unité de présentation n°1

Nous avons identifié 5 sous-tâches que nous décrivons dans les 5 Unités de Présentations (UP) suivantes. Chaque Unité de Présentation est composée d'un certain nombre de fenêtres. Notre regroupement en UP et en fenêtre est basé soit sur un critère fonctionnel soit sur un critère personnel (c'est-à-dire le type de personnes concernées par la sous-tâche).

Dans l'UP1, nous avons identifié trois fenêtres. La première (fenêtre 1) permet la connexion du personnel médical à la carte du patient. La seconde (fenêtre 2) invite le médecin, l'infirmière ou l'hôtesse à s'authentifier. La dernière (fenêtre 3) signifie à celui-ci qu'il s'est authentifié d'une façon erronée.

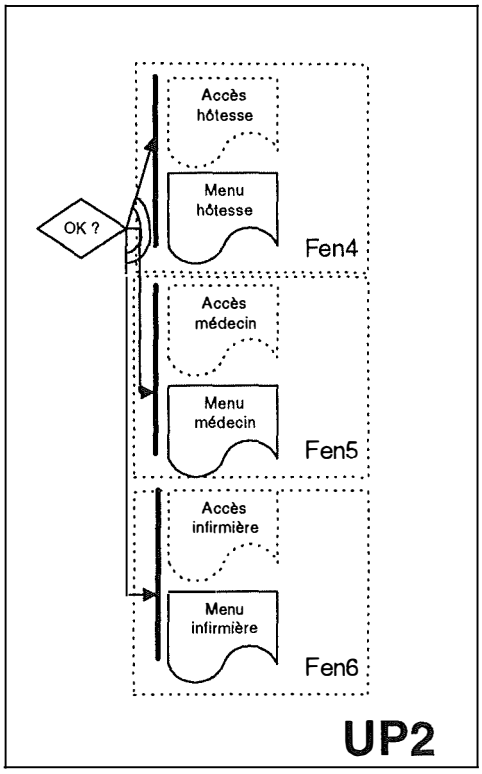


Figure 32 : L'unité de présentation n°2

L'Unité de Présentation 2 regroupe les différents choix proposés dans les menus contextuels du personnel médical. Le critère personnel est très clairement appliqué pour le regroupement fonctionnel. La fenêtre 4 correspond aux opportunités de l'hôtesse. Les médecins consultent les informations concernant le patient par la seconde fenêtre de cette UP (fenêtre 5). Le dernier menu (fenêtre 6) est celui de l'infirmière.

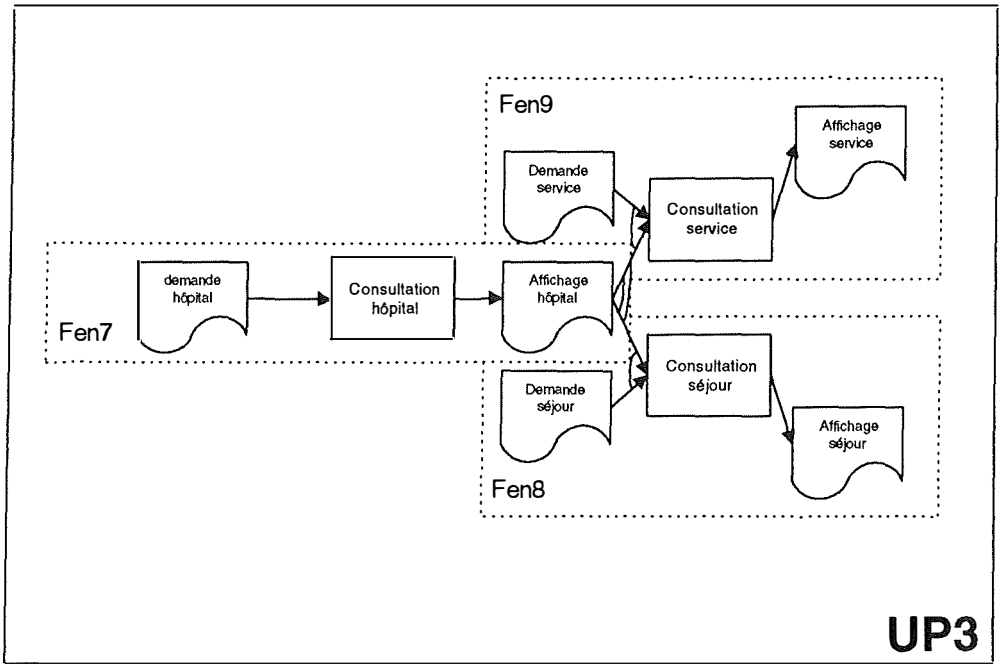


Figure 33 : L'unité de présentation n°3

L'UP 3 est composée de 3 fenêtres et elle concerne plus particulièrement l'hôtesse. La première (fenêtre 7) sert à consulter les hôpitaux dans lesquels le patient a déjà séjourné. L'aperçu des séjours effectués par le patient dans un hôpital est présenté dans la fenêtre 8. La dernière fenêtre de cette UP 3 (fenêtre 9) est utile pour gérer les services d'un hôpital.

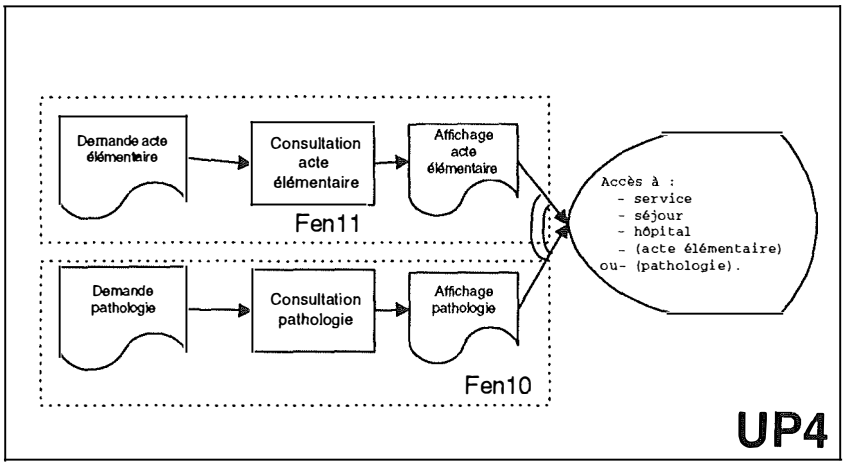


Figure 35 : L'unité de présentation n°4

L'unité de présentation 4 comporte deux fenêtres. Cette UP concerne uniquement le médecin. La fenêtre 10 permet aux médecins de s'informer au sujet d'une pathologie dont souffre le patient. Le médecin peut s'enquérir des actes élémentaires subis par le patient grâce à la fenêtre 11.

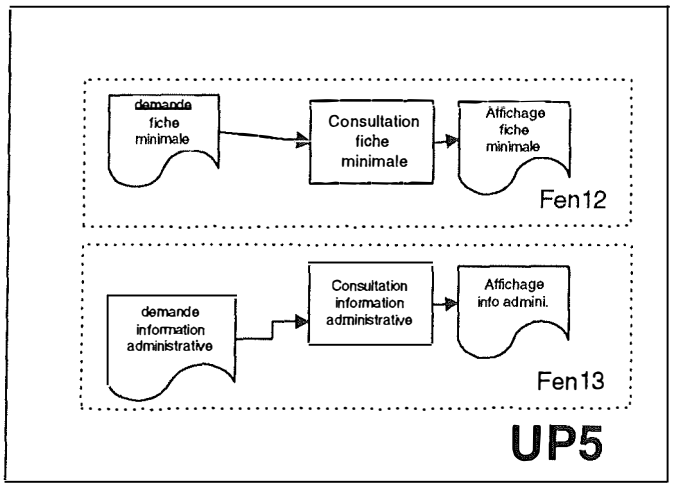


Figure 34 : L'unité de présentation n°5

Nous avons divisé l'unité de présentation 5 en 2 fenêtres. Cette UP est composée de 2 fonctions qui sont offertes à tous les membres hospitalisés. La fiche minimale du patient est visualisable par la fenêtre 12. La dernière fenêtre (fenêtre 13) présente les informations administratives concernant le patient.

5. Implémentation

5.1. L'environnement

Les CGI ont été développés en **C++ sous Unix**. Dans les CGI, que nous avons développés, l'utilisation du paradigme objet est plutôt faible. Le C++ s'imposait néanmoins pour des raisons de compatibilité avec la plate-forme Orbix.

Un total de **39 CGI** a été écrit. D'une manière générale, les CGI consistent en l'envoi d'une ou plusieurs requêtes à la carte et en l'affichage des résultats. Pour nous faciliter la tâche, nous avons développé une vingtaine de fonctions qui étaient communes à plusieurs CGI.

5.2. Orbix

Orbix est une implémentation particulière de la norme CORBA. Nous allons brièvement expliciter le fonctionnement général de CORBA [OHE96].

5.2.1. Révolution dans le monde Client/Serveur

Les objets distribués vont permettre de décomposer les grosses applications mono-bloc en composants (objets) distribués sur le réseau qui coopèrent entre eux. On peut voir les nouvelles applications Client/Serveur comme constituées de composants qu'on assemble pour les rendre tout à fait adaptées aux besoins des utilisateurs.

5.2.2. Objets distribués et composants

Les **objets classiques** sont des entités intelligentes qui encapsulent des données et des fonctions (méthodes) qui permettent de les manipuler. Ils offrent une grande réutilisabilité du code par le concept d'héritage. Les objets classiques vivent dans un seul programme : le monde extérieur au programme n'a pas connaissance de l'existence des objets.

Un **objet distribué** est une entité intelligente qui vit quelque part sur le réseau : des clients peuvent y accéder en invoquant ces méthodes. Le client n'a pas besoin de savoir où se trouve l'objet, par quel réseau il faut y accéder, sur quel système d'exploitation il est exécuté, par quel langage il a été créé et par quel compilateur il a été compilé : tout ceci est transparent pour le client.

Les composants permettent de créer des applications Client/Serveur très vite en assemblant et en étendant (par le mécanisme d'héritage) les composants. Ils vont bouleverser la manière dont les applications sont construites et distribuées :

- les utilisateurs pourront personnaliser leurs applications en y intégrant les composants dont ils ont besoin. Actuellement, l'utilisateur est contraint d'acheter l'entièreté d'une application (Word, Excel, ...) alors qu'il n'utilise souvent pas plus de 10% des fonctionnalités de celle-ci : les autres fonctionnalités ont pour seul effet d'ajouter de la complexité. Avec les composants, l'utilisateur pourra acquérir uniquement les parties de l'application dont il se sert.
- les petits développeurs pourront utiliser les composants offerts par les gros distributeurs pour les étendre et fournir de nouvelles applications.
- les grands développeurs utilisent les composants pour créer (ou assembler) des applications Client/Serveur en un temps record. Les applications étant constituées de blocs indépendants qui coopèrent entre eux, elles deviennent plus faciles à coder, débugger et tester.
- les vendeurs utilisent les composants pour assembler des applications qui visent des marchés aux besoins spécifiques. On construit par exemple un Word parfaitement adapté aux exigences d'un cabinet d'avocat.

5.2.3. Corba, the Distributed Object Bus

CORBA (*Common Object Request Broker Architecture*) est fondamentalement un bus qui permet aux composants qui y sont connectés de communiquer et de collaborer entre eux à travers les régions, pays et continents. CORBA est un standard développé par un consortium de plus de 500 entreprises du monde de l'informatique et des télécommunications regroupées sous le sigle OMG (*Object Management Group*). Il existe actuellement une douzaine d'implémentations de CORBA 1.2 sur le marché; Orbix en est une.

a) Aperçu général

Dans CORBA, un composant se définit comme une entité intelligente, vivant sur le bus, à laquel les clients peuvent accéder en invoquant ses méthodes. Le client n'a pas à se soucier de la localisation de l'objet dans le réseau, de la manière dont l'objet est implémenté (langage, compilateur), du système d'exploitation sur lequel l'objet est exécuté ou des réseaux qu'il faut traverser pour y accéder.

La seule chose que le client doit connaître pour accéder à un composant, c'est son interface : elle sert de lien entre le client et le serveur (c'est une espèce de contrat entre les deux parties). L'interface d'un objet est décrite dans un langage de spécification standard : IDL (*Interface Definition Language*).

IDL ne contient aucun détail d'implémentation : il est purement déclaratif. Il permet de décrire les caractéristiques des composants, les classes parents dont il dérive, les erreurs qu'il produit, le type d'évènement qu'il émet et les méthodes qu'il propose (avec la description des paramètres à fournir en entrée et la description des résultats). Tout composant qui vit sur le bus ORB possède une interface écrite en IDL : c'est cette interface qui permet à des composants écrits en langages différents et exécutés sur des systèmes hétérogènes de communiquer entre eux.

b) Evolution sémantique des objets

Comme nous l'avons déjà signalé à plusieurs reprises, dans le monde des objets distribués, **les applications deviennent une collection de composants qui communiquent entre eux**. L'objectif ultime (le Nirvana dans le monde des objets distribués) est d'arriver à des composants qui ne se contentent pas de communiquer mais qui coopèrent entre eux au niveau sémantique pour effectuer une tâche : ces composants sont appelés *Business Objects*.

La *Figure 36* montre l'évolution des composants de la simple communication (interopérabilité) jusqu'à la collaboration sémantique.

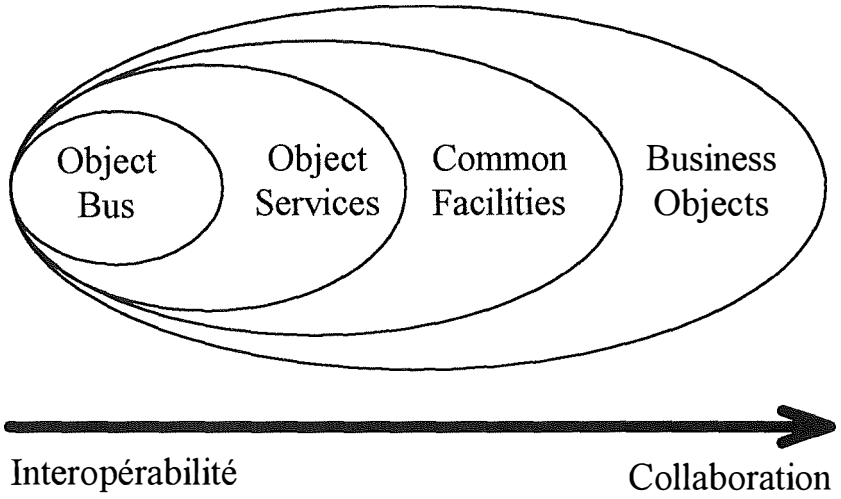


Figure 36 : L'évolution sémantique des composants

L'élément le plus fondamental est le bus ORB de CORBA (*object bus*) : il permet aux composants de communiquer entre eux par invocation des méthodes.

L'étape sémantique suivante est celle des services nécessaires à tous les composants (*object services*) : ces services permettent notamment de nommer les composants, de gérer les accès concurrents à un même composant, de gérer la sécurité (tout le monde n'a pas accès à un composant), de gérer les licences (quand un client utilise un composant, il doit payer le concepteur de l'objet), etc. Ces services sont proposés dans l'architecture de CORBA et sont communs à tous les composants : chaque composant placé sur le bus utilise ou non ceux-ci en fonction des besoins.

Les *Common Facilities* fournissent les standards de collaboration entre composants sous la forme de squelette d'application (application framework). Elles permettent d'assembler certains composants au sein d'une même application en fournissant les fondations de celle-ci. Les squelettes d'application fixent les règles qui régissent la manière dont les composants vont travailler ensemble : ils permettent aux composants de collaborer dans une application.

Au niveau sémantique le plus haut, on retrouve finalement les *Business Objects*. Il s'agit de composants qui modélisent une partie du monde réel. On peut par exemple imaginer un *Business Object* qui représente une personne (qui **nous** représente), une entreprise, un hotel, ... Au niveau des *Business Objects*, les composants collaborent entre eux pour éxécuter une tâche : par exemple, une personne commande un produit à une entreprise ou réserve une nuit dans un hotel. C'est plus que de la simple communication.

A partir de cette évolution sémantique des composants, on définit l'architecture de CORBA.

c) Architecture de CORBA

L'architecture de CORBA est constituée de 4 éléments fondamentaux :
- *Object bus ORB* (*Object Request Broker*)
- *Object Services*
- *Common Facilities*
- *Application Objects* (*Business Objects*)

L'architecture globale est représentée par la *Figure 37*.

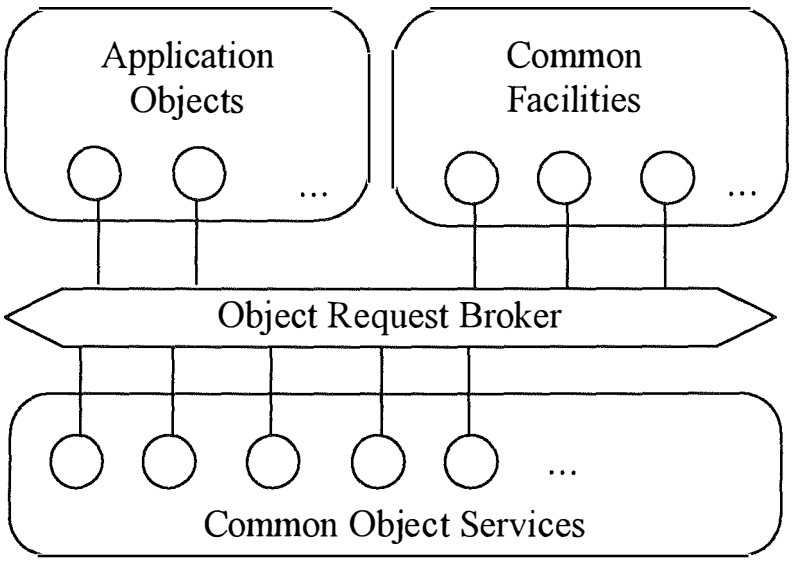


Figure 37 : L'architecture de CORBA

Object Request Broker

Comme nous l'avons déjà signalé, ORB est le bus de CORBA. Celui-ci offre les fonctionnalités suivantes :
- invocation statique et dynamique : les méthodes qui sont invoquées par un client sont définies statiquement au moment de la compilation ou elles sont découvertes dynamiquement à l'exécution. L'invocation dynamique permet aux objets de se découvrir l'un l'autre.

- lien avec langage de haut niveau : on peut invoquer les méthodes d'un composant du bus à partir de n'importe quel langage. Tous les composants du bus sont indépendants du langage dans lequel les autres composants ont été écrits. Ce qui importe, c'est l'interface d'un composant (spécifiée en IDL).
- system auto-descriptif : chaque bus ORB dispose de toutes les méta-données qui décrivent (en IDL) les interfaces des composants qui sont sur ce bus : ces interfaces sont regroupées dans un *Interface Repository*.
- transparence de localisation : lors de l'invocation des méthodes d'un objet, le client n'a pas à se soucier de la localisation de l'objet sur le bus. C'est ORB qui se charge d'accéder à l'objet : tout est transparent pour le client.
- polymorphisme : lorsqu'on invoque une méthode, on n'invoque pas seulement une fonction mais on invoque une fonction sur un objet spécifique. Ainsi, une fonction peut avoir des effets différents suivant l'objet sur lequel la fonction est exécutée.

Object Services

L'objectif des *Objects Services* est d'augmenter les fonctionnalités de ORB. Les *Object Services* sont des composants reliés au bus. Parmis ces services, on retrouve entre autre :

- *Life Cycle Service* : permet de créer, copier, déplacer et effacer un composant sur le bus.
- *Naming Service* : permet de nommer les composants et d'y accéder par leur nom.
- *Concurrency Control Service* : permet de gérer les accès concurrents à un même composant.
- *Licensing Service* : permet au concepteur d'un objet de toucher une rémunération de la part des clients qui l'utilisent.
- *Security Service* : permet de contrôler l'accès à un objet.

Dans CORBA, les fournisseurs de composants peuvent développer ceux-ci sans se soucier des *Object Services*. Une fois les composants construits, le développeur peut associer son composant à un service (qui est aussi un composant) en créant un objet de plus haut niveau (objet générique) qui hérite des deux objets initiaux (composant de base et *Object Service*) : on utilise l'héritage multiple. Le développeur associe son composant à des services en fonction des besoins : il l'associera, par exemple, au *Naming Service* et au *Concurency Control Service* mais pas au *Licensing Service* ni au *Security Service* parce qu'il estime que tout le monde peut accéder à son composant sans devoir payer quoi que se soit.

Common Facilities

Il s'agit de composants qui offrent des squelettes d'application (*application framework*). Ces composants permettent de construire des applications complètes (qui deviennent à leur tout des composants du bus) en assemblant des composants de base indépendants comme des Légos. Les squelettes définissent comment les composants vont collaborer au sein de l'application.

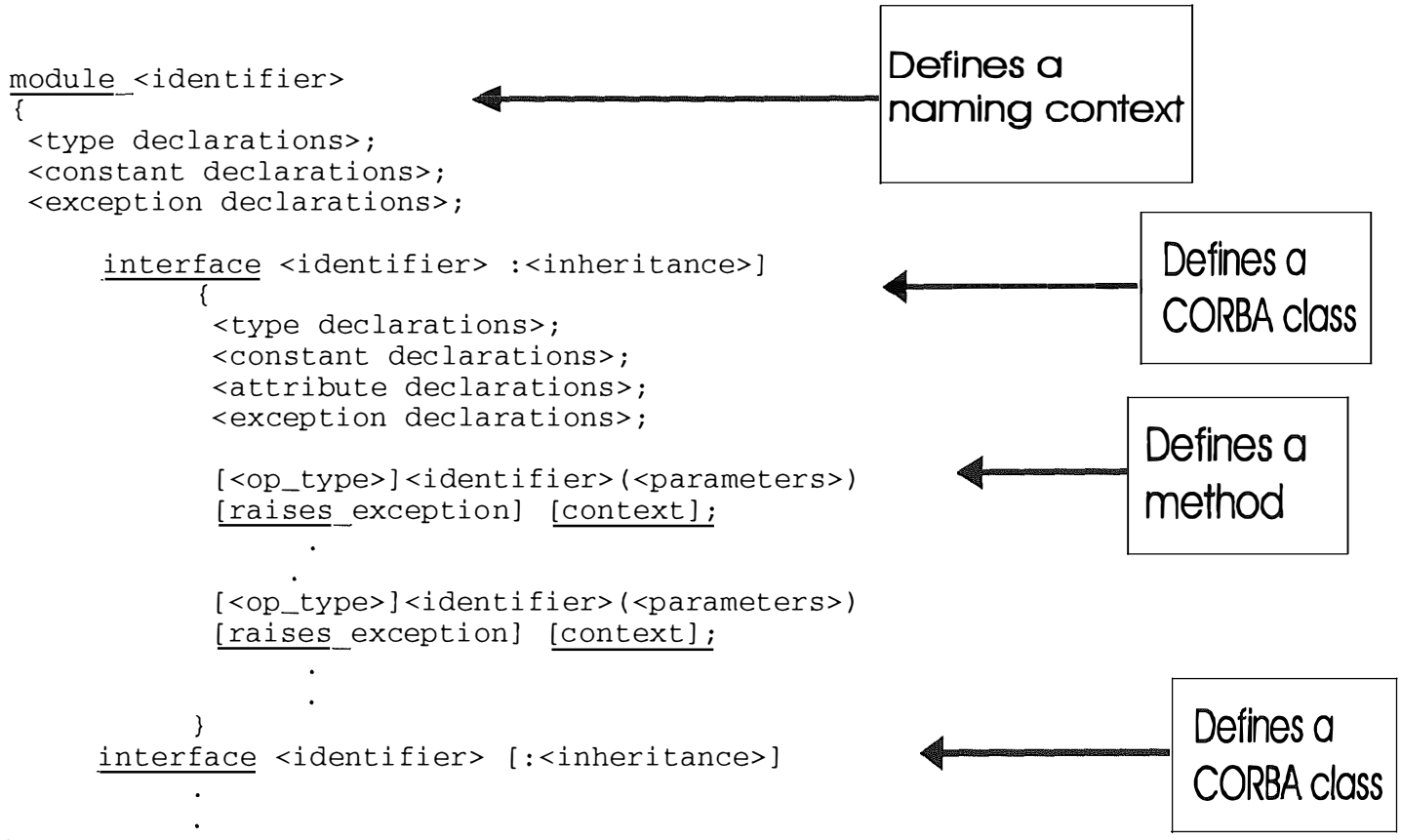
Application/Business Objects

Ces composants correspondent aux applications *End-User* construites en assemblant des composants indépendants appartenant aux trois niveaux précédants.

5.2.4. The Interface Definition Language (IDL)

Corba contient des méta-données. Une méta-donnée est une information auto-descriptive. Le langage de méta-données de Corba s'appelle Interface Definition Language (IDL). IDL est un langage qui permet de spécifier les frontières d'un composant et son interface avec les clients potentiels. IDL est un langage neutre et totalement déclaratif : il ne contient aucun détail d'implémentation. IDL fournit des interfaces indépendantes de tout langage de programmation et de tout système d'exploitation aux services et composants qui résident sur le bus Corba.

a) La structure d'un fichier IDL



- Un module comprend le nom donné à un ensemble de descriptions de classes.

- Une interface est un ensemble de méthodes que le client peut appliquer sur un objet. C'est comme une définition de classe sans l'implémentation. Dans une interface, on peut déclarer une ou plusieurs exceptions qui sont des messages indiquant qu'une opération n'a pas eu lieu correctement. Une interface peut également disposer d'attributs.

- Les opérations sont les équivalents dans le langage Corba des méthodes. Un client pourra les appliquer sur un objet. On parle de *signature* de la méthode à propos des

types des paramètres et du résultat. Un paramètre contient un *mode* qui indique si le paramètre est passé du client au serveur (*in*), du serveur au client (*out*) ou bien les deux (*inout*). On peut également retrouver un retour d'erreur par l'option *raises exceptions*. Enfin, un *contexte* peut contenir un ensemble d'attributs qui représentent le contexte du client. Le client peut passer des informations au serveur par ce moyen.

- Les types de données supportés par Corba sont de deux catégories : de base et construits. Les types de base sont : short, long, unsigned long, unsigned short, float, double, char, boolean and octet. Les types construits sont : enum, string, struct, array, union, sequence and any. Le type any est très utile pour les déclarations de type dans le cadre dynamique car il peut représenter tous les types.

b) Le fichier IDL

Dans notre application, **un objet serveur de carte a été implémenté**. Dans la première figure du chapitre I, nous avons expliqué qu'un Client/Serveur a été mis en place entre le lecteur de carte du Client carte et le Serveur. En effet, une connexion doit être maintenue en permanence entre le lecteur de carte et la machine à laquelle il est directement relié.

L'objet serveur de carte joue ce rôle de processus qui tourne en permanence et qui attend les clients. Il est situé du côté du Client carte (c'est-à-dire là où se trouve le lecteur de carte). L'objet serveur de carte propose une seule méthode qui est décrite dans le fichier simu.idl. Lorsqu'un client applique la méthode ToCard à un objet de type simu, il place comme paramètre en entrée la requête qu'il désire que la carte exécute. Le serveur envoie au client le résultat et des informations concernant l'état de l'exécution.

c) / simu.idl */*

```
interface simu {

     void ToCard(in char method[255], out char result[2048],
                 out char info[255]);

};
```

5.3. Un modèle de CGI

A titre d'illustration, nous vous présentons un exemple de CGI. Pour des raisons de confidentialité, vous ne trouverez pas le listing de notre application en annexe. La firme Gemplus est susceptible d'utiliser notre poduit et ne tient donc pas à divulguer l'ensemble des sources.

Dans la suite, nous expliquons au fur et à mesure que nous les rencontrons les principales instructions. Celles-ci se retrouvent d'une façon générale dans la plupart des CGI que nous avons écrits.

```
#include <iostream.h>
#include "simu.hh"
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "util2.h"
#include "util.h"

void aelem2::main(int argc, char *argv[])
{
char cmdin[256],*cmdout,*cmdout2,*info,*cl=NULL,
data[65],ch2[3];
int  i=1, pasdeboucle=0, seq[3];
simu*s;
```

/* les printf envoient sur la sortie standard un ensemble de caractères. Dans notre cas, **le CGI construit la page HTML qui sera envoyée au client**. Cette première ligne spécifie au client que le texte qui lui est envoyé est de type HTML.. Les instructions printf contiennent des éléments qui génèrent la page HTML et le contenu de ces instructions respectent le format HTML présenté dans le point 3 du chapitre III. */

```
     printf("Content-type: text/html%c%c",10,10);
```

/* les lignes qui suivent permettent à l'objet Orbix s de se connecter à Orbix */

```
     TRY {
          s = simu::_bind("",getenv("ORBIX"), IT_X);
     } CATCHANY {
    cerr << "Bind to object failed" << endl;
    cerr << "Unexpected exception " << IT_X << endl;
          return ;
  } ENDTRY
```

/* la variable cl comprend le QUERY_STRING c'est-à-dire les caractères qui suivent le '?' de l'URL. Comme nous l'avons expliqué dans le point 4.2 du chapitre III, il existe deux types de CGI : Les POST et les GET. Dans ce cas, nous sommes typiquement face à une méthode GET. C'est pourquoi nous récupérons les données contenues dans l'URL. */

```
     cl = getenv("QUERY_STRING");
```

```
if (!(cl == NULL || strcmp(cl,"aelem")==0)) pasdeboucle=1;
```

/* Nous avons créé une fonction qui génère le texte HTML pour le titre général du document et le titre de la page */

```
SetTitre ("Liste des actes elementaires","Actes
          élémentaires du patient","1");
```

/* Nous indiquons au browser que le CGI que nous sommes en train de créer est de type GET. */

```
printf ("<METHOD=GET>");
```

```
/* Dans ch2, nous plaçons le numéro de l'acte élémentaire que
nous recherchons. Celui-ci a été fourni par le CGI précédent.
*/
```

```
ch2[0]=cl[6];
if (isdigit(cl[7])!=0)
{
     ch2[1]=cl[7];
     ch2[2]='\0';
}
else
     ch2[1]='\0';
```

/* Dans le cas où la variable pasdebloucle vaut 1, cela signifie que l'on désire rechercher une pathologie précise du patient. Dans le cas contraire, on recherche l'ensemble des pathologies dont a souffert le patient. */

```
if (pasdeboucle==1)
{
     sprintf(cmdin,"select * from AE where
     NAE='%s';",ch2);
}
else
{
     strcpy(cmdin,"select * from AE;");
}
```

/* Dans la variable cmdin, nous avons placé la requête SQl qui sera soumise à la carte. C'est la méthode **ToCard qui envoie cette requête à la carte**. La carte exécute celle-ci et la méthode ToCard récupère les résultats et les informations concernant l'exécution de la requête. */

```
s->ToCard(cmdin, cmdout, info) ;
```

/* La fonction GetCqlOut(i,j,cmdout,data) insère dans data le contenu de la jème colonne de la ième ligne de la variable cmdout (c'est-à-dire le résultat de la requête). La fonction renvoie 1 si tout se passe bien. */

```
while (GetCqlOut(i,1,cmdout,data)==1)
{
```

/* insère le nom de l'acte élémentaire */

```
if (GetCqlOut(i,2,cmdout,data)==1)
     printf("<LI><H2><I>%s</I>", data);
```

/* insère la date à laquelle l'acte a été posé */

```
if (GetCqlOut(i,3,cmdout,data)==1)
     printf(" effectué(e) le %s", data);
printf("</H2></LI><H3>");

printf("<BLOCKQUOTE>");
if (GetCqlOut(i,4,cmdout,data)==1)
     printf("Type d'acte : %s<BR>", data);
if (GetCqlOut(i,5,cmdout,data)==1)
     printf("Compte-rendu : %s<BR>", data);
```

/* data contient le numérro du service qui a effectué l'acte. */

```
GetCqlOut(i,6,cmdout,data);
if (data[0]!='R')
{
```

/* On place dans cmdin la requête qui recherche les informations concernant ce service. */

```
     strcpy(cmdin,"select NOM from SER where
     NSER='");
     strcat(cmdin,data);
     strcat(cmdin,"';");
     s->ToCard(cmdin, cmdout2, info) ;
```

/* La procédure Config insère une ancre dans le document HTML vers le service. Elle permet également d'insérer un élément de paramétrage. */

```
     Config ("<BR>Acte effectué dans le service : \
          <A HREF=\"http://","/cgi-
          bin/weblink?service+m+");
     printf("%s\">",data);
     if (GetCqlOut(1,1,cmdout2,data)==1)
          printf("%s</A><BR>", data);
     else
          printf("cliquez ici</A><BR>");
     printf("</BLOCQUOTE>");
}
GetCqlOut(i,1,cmdout,data);
seq[0]=4;
seq[1]=6;
```

/* Comme nous l'avons expliqué dans le point 3.4 de ce chapitre, à la table des actes élémentaires est associée une table des références. La procédure RefPlusieurs3 effectue une requête pour trouver les références des actes élémentaires. Elle affiche des ancres vers les références. */

```
            RefPlusieurs3("*","RAE","NAE",data,"Références de
                              l'acte :",seq);
            printf("</BLOCKQUOTE></BLOCKQUOTE><BR>");
```

/* Si l'on ne cherche qu'une pathologie, on quitte la boucle. Dans le cas contraire, on poursuit la recherche. */

```
            if (pasdeboucle == 1)
            {
                 break;
            }
            else
            {
                 i++;
            }
   }
```

/* Dans le cas où l'on effectue une recherche exhaustive des actes élémentaires posés sur le patient, on peut insérer de nouveaux actes ou en modifier. */

```
      if (pasdeboucle!=1)
      {
           Config("<CENTER><H3><I><A HREF=\"http://",\
          "/cgi-bin/weblink?maelem+inserer+m\">Insérer un
           acte</A><P>");
          Config("<A HREF=\"http://"\
          ,"/cgi-bin/weblink?maelem+modifier+m\">Modifier un
           acte</A></I>");
      }
      printf("</BLOCKQUOTE>");
```

/* La procédure RetourPrincipal insère une ancre vers le menu principal du médecin, de l'infirmière ou de l'hôtesse. */

```
      RetourPrincipal('R');
```

/* La procédure RetourConnect insère une ancre vers le menu de connexion du patient. */

```
      RetourConnect();
```

/* La déconnexion de Orbix. */

```
   s->_release() ;
}
```

5.4. L'implémentation de la base de données en CQL

Ce paragraphe décrit le code CQL qui est attaché à la création de la structure de tables décrites dans la *Figure 25*. Notons de plus que les contraintes d'intégrité du type identifiant, clé étrangère, ... ne sont pas pris en compte par le gestionnaire CQL. Les programmes d'application devront donc gérer ces mécanismes par eux-mêmes. La matrice d'accès du *Tableau 6* est implémentée par l'ensemble des grant situés à la fin du fichier.

```
present E 'password';
create application CHRU 3 'CHRU';
present 'CHRU' CHRU;
create user HOT 3 'HOTPASS';
create user INF 3 'INFPASS';
create user MED 3 'MEDPASS';
create table ADMIN(NOM,PRE,DDN,ADR,TEL,NSS,HMP);
create table AE(NAE,INT,DATE,TYPE,CR,NSER,NP);
create table ALLG(TYPE,TRAIT,CTRI,COM);
create table FMIN(GS,ANT,COM);
create table HOP(NHOP,ADR);
create table PASEJ(NP,NS);
create table PAT(NP,INT,DATE,NSER);
create table RAE(NAE,TYPE,TEXT,REF1,REF2,REF3,REF4);
create table RHOP(NHOP,TYPE,TEXT,REF1,REF2,REF3,REF4);
create table RPAT(NP,TYPE,TEXT,REF1,REF2,REF3,REF4);
create table RSEJ(NS,TYPE,TEXT,REF1,REF2,REF3,REF4);
create table RSER(NSER,TYPE,TEXT,REF1,REF2,REF3,REF4);
create table SEJ(NS,DIN,DOUT,NHOP);
create table SER(NSER,NOM,ADR,SPE,CHEF,NHOP);
create table VACC(TYPE,DATE,REAC);
present CHRU 'CHRU';
grant select on ADMIN to MED;
grant select on ADMIN to INF;
grant all on ADMIN to HOT;
grant all on AE to MED;
grant all on ALLG to MED;
grant select on ALLG to INF;
grant select on ALLG to HOT;
grant insert on ALLG to HOT;
grant all on FMIN to MED;
grant select on FMIN to INF;
grant select on FMIN to HOT;
grant insert on FMIN to HOT;
grant select on HOP to MED;
grant select on HOP to INF;
grant all on HOP to HOT;
grant all on PASEJ to MED;
grant all on PAT to MED;
grant all on RAE to MED;
grant select on RHOP to MED;
grant select on RHOP to INF;
grant all on RHOP to HOT;
grant all on RPAT to MED;
grant all on RSEJ to MED;
grant all on RSEJ to HOT;
grant select on RSER to MED;
grant insert on RSER to MED;
grant all on RSER to HOT;
```

```
grant all on SEJ to MED;
grant all on SEJ to HOT;
grant all on SER to MED;
grant all on SER to HOT;
grant all on VACC to MED;
grant select on VACC to INF;
grant select on VACC to HOT;
grant insert on VACC to HOT;
```

Nous proposons un exemple de requêtes qui permet d'illustrer l'insertion des informations médicales concernant un patient dans sa carte. Notre application carte en fonction de la classification décrite au point 2.5 du chapitre II entre dans la catégorie des dossiers portables. En effet, elle contient des informations sur le porteur.

```
insert into ADMIN values ('Pierre','Dupont','10/04/66',
'','','','');
update ADMIN set ADR='10 rue du Framboisier';
insert into ALLG values ('acarien','cortison','poussière','fini');
insert into FMIN values ('O+','appendicite','sang liquide');
insert into VACC values ('fièvre jaune','22/04/92','fièvre 2
jours');

insert into HOP values ('1','CHRU Lille');
insert into RHOP values ('1','http','home
page','','','index.html','');

insert into SEJ values ('1','29/03/94','21/04/94','1');
insert into RSEJ values ('1','avant','','','jambe.gif');
insert into SER values ('1','Cardiologie','Bat.2','greffe',
'Paradinas','1');

insert into PAT values ('1','Herpes','12/10/93','1');

insert into AE values ('1','consultation','22/08/94','','Herpes',
'1','1');
insert into RAE values ('1','http','diagnostic','dg7.html',
'','','');
```

insert into PASEJ values ('1','1');

5.5. Des exemples de fenêtres

Afin de clôturer ce chapitre, nous vous présentons un scénario de navigation possible dans notre application et nous y associons les fenêtres.

Un médecin se connecte à la page web du serveur (*Figure 38 : Le menu de connexion*).

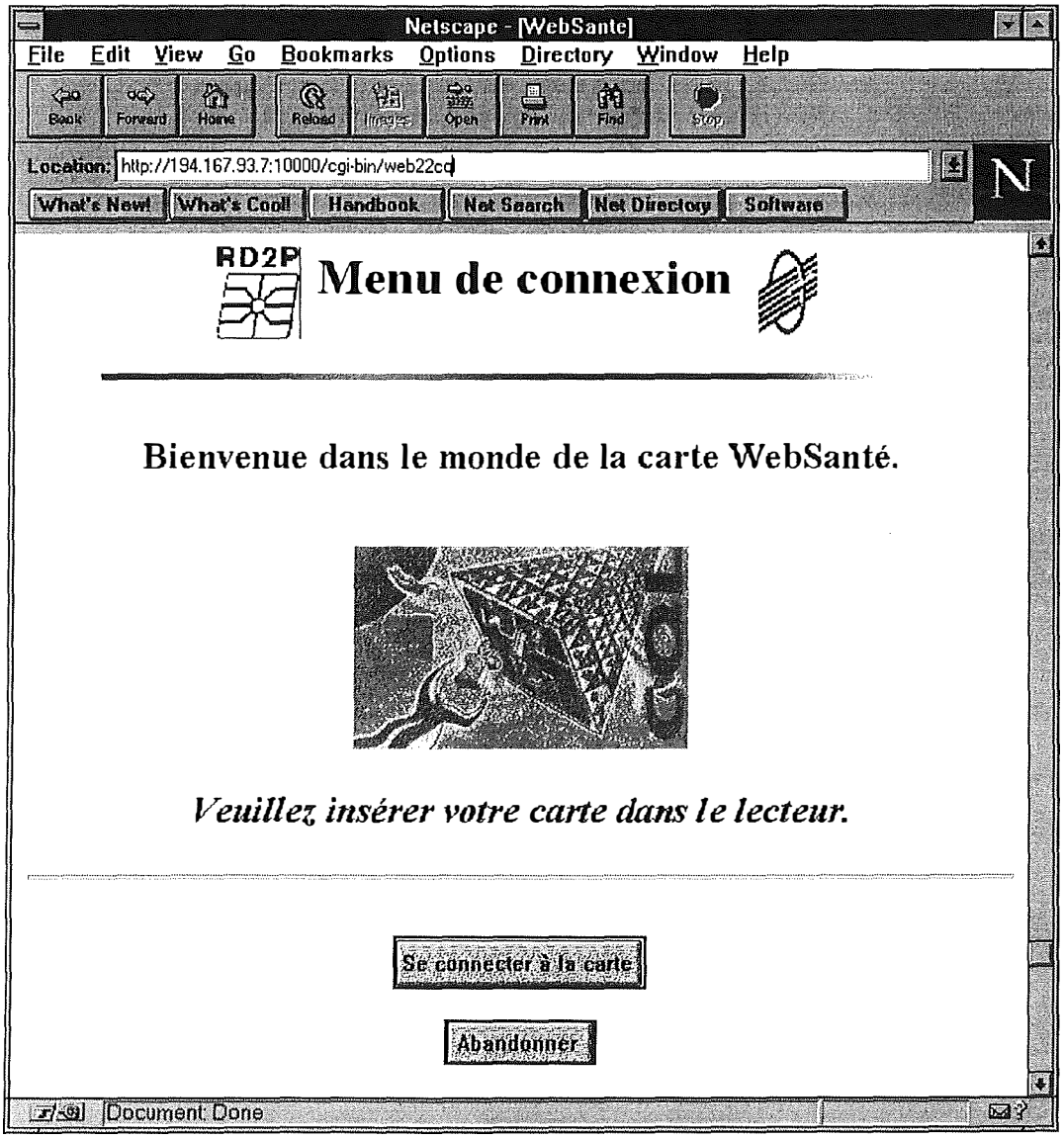


Figure 38 : Le menu de connexion

Le médecin choisit de lire des informations sur la carte. Une fenêtre d'identification et d'authentification l'invite à introduire son mot de passe (*Figure 39 : Le menu d'authentification*).

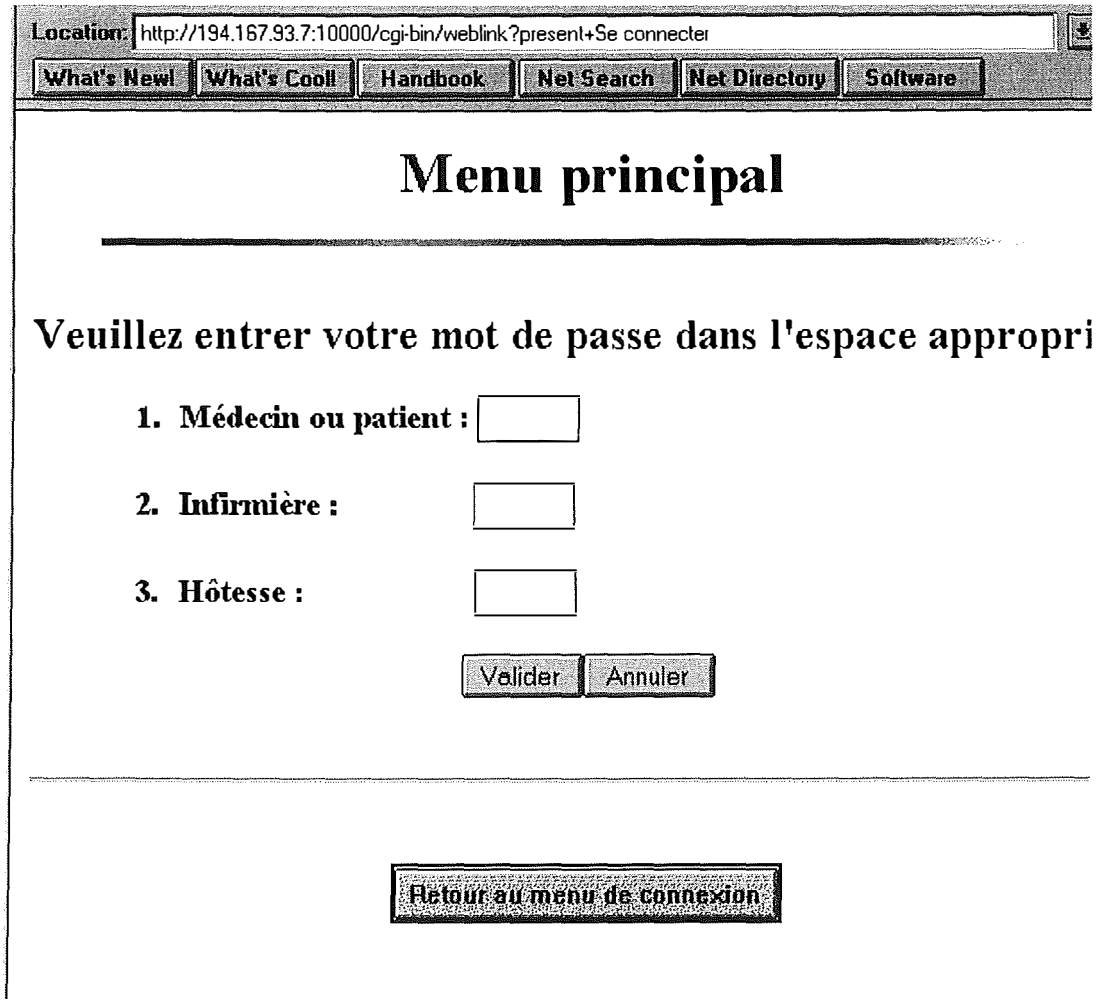


Figure 39 : Le menu d'authentification

Il est alors averti de l'exactitude de son mot de passe (*Figure 40 : Le menu de confirmation de l'authetification*).

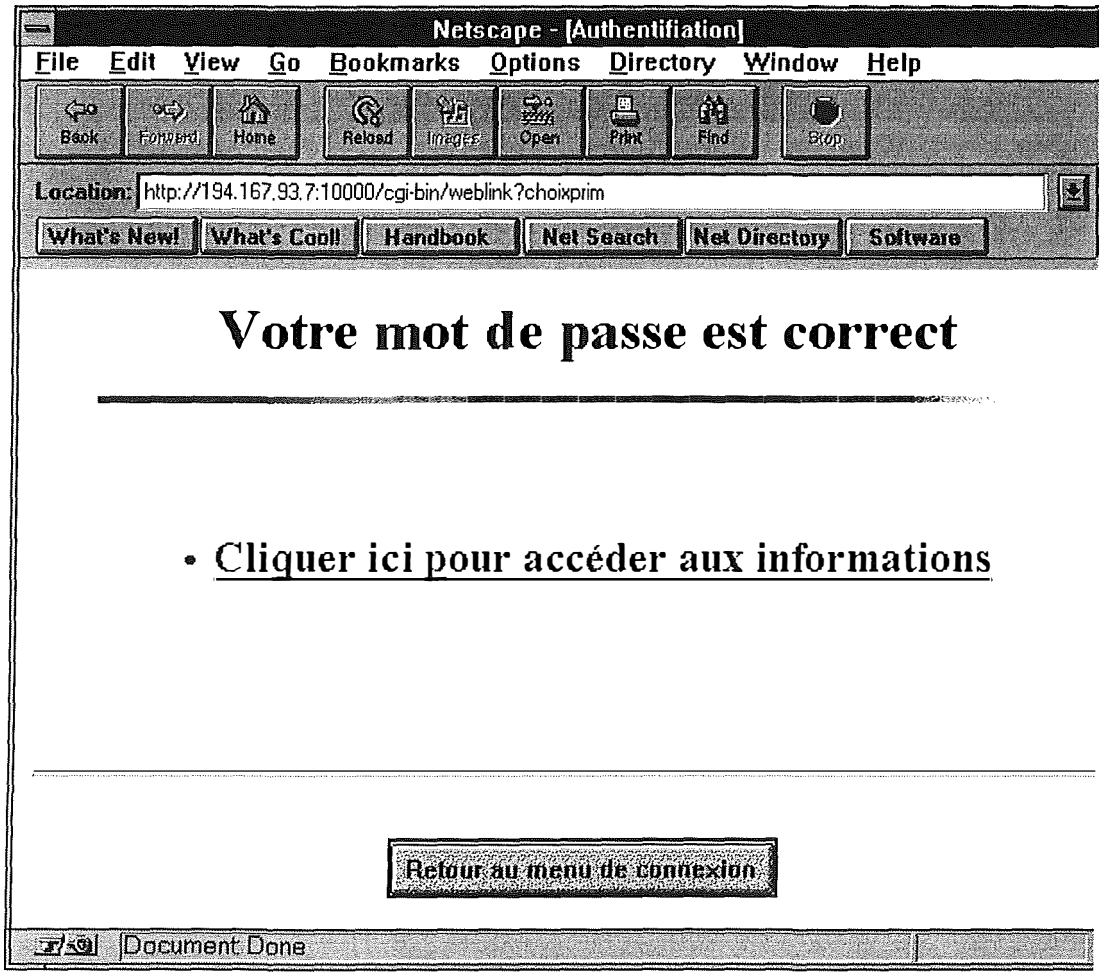


Figure 40 : Le menu de confirmation de l'authetification

Ensuite, le médecin voit apparaître à l'écran un menu (*Figure 41 : Le menu du médecin*). Il souhaite prendre connaissance des pathologies dont souffre son patient. A cet effet, il choisit le quatrième point du menu.

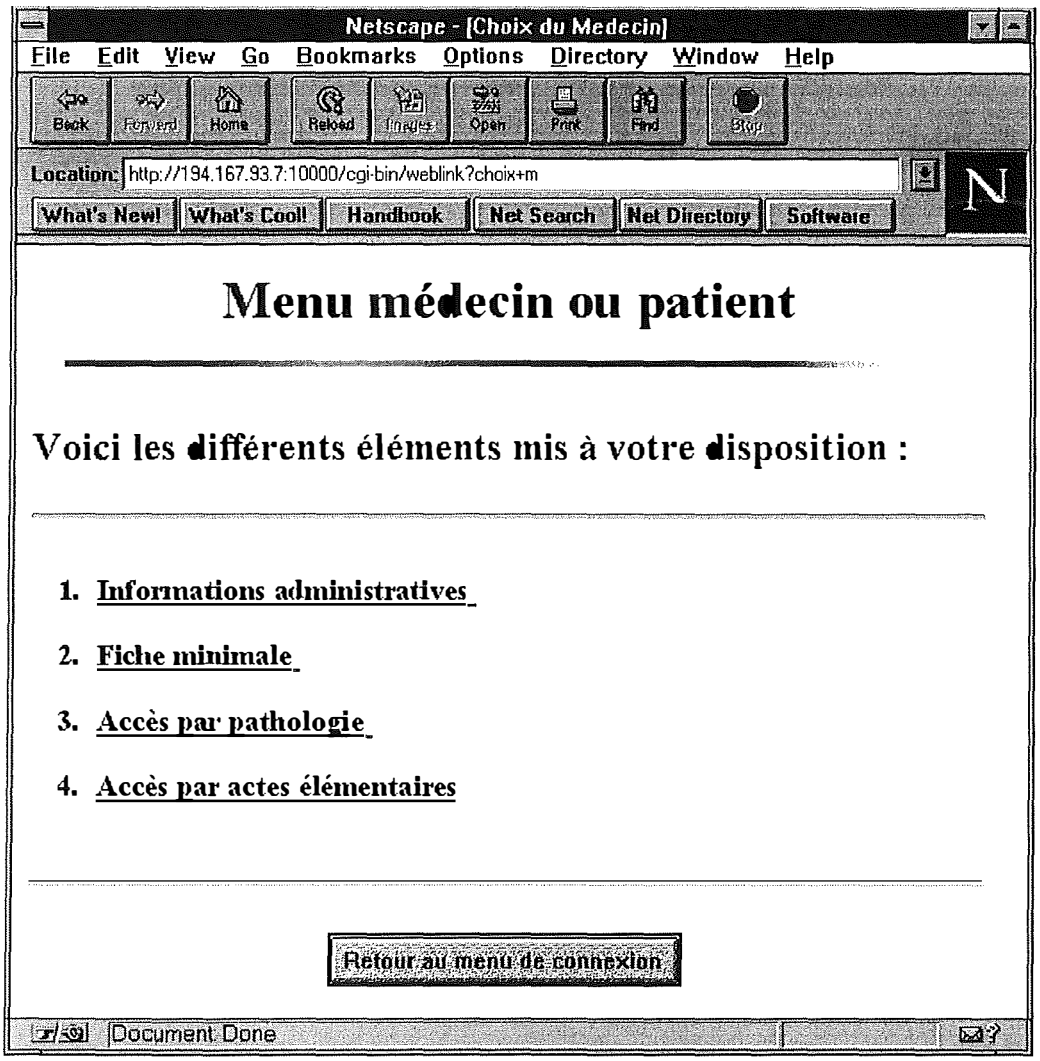


Figure 41 : Le menu du médecin

La fenêtre (***Erreur! Source du renvoi introuvable.***) affichant les pathologies du patient est présentée au docteur. Il désirerait en savoir un peu plus sur la première pathologie du patient : l'Erysipel.

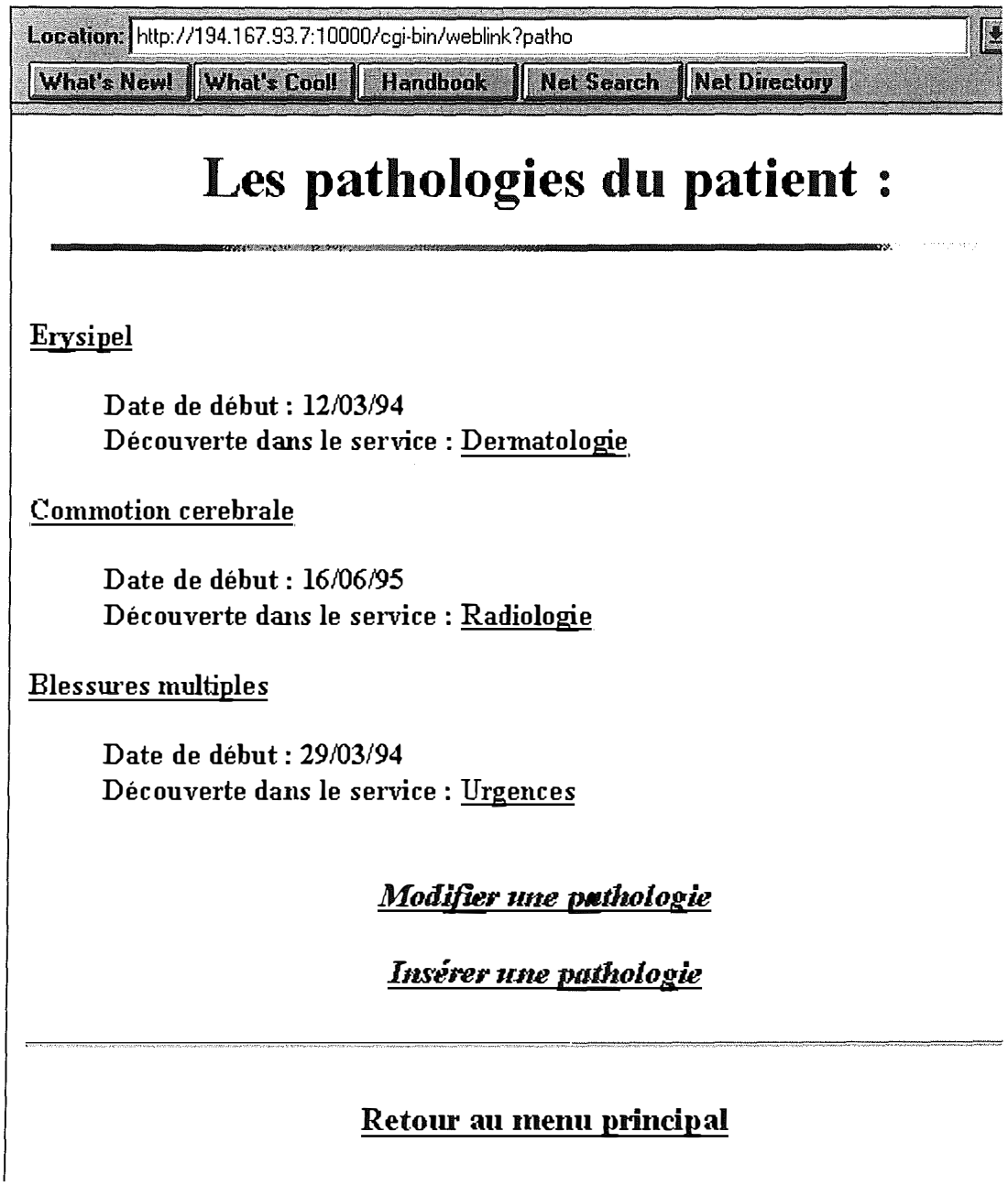


Figure 42 : Les pathologies d'un patient

La fenêtre (*Figure 43 : Une pathologie du patient*) fournit les informations suivantes : la date de découverte de la pathologie, les séjours effectués dans ce cadre, les actes élémentaires, l'hôpital dans lequel les actes ont été entrepris, le service dans lequel la pathologie a été découverte.

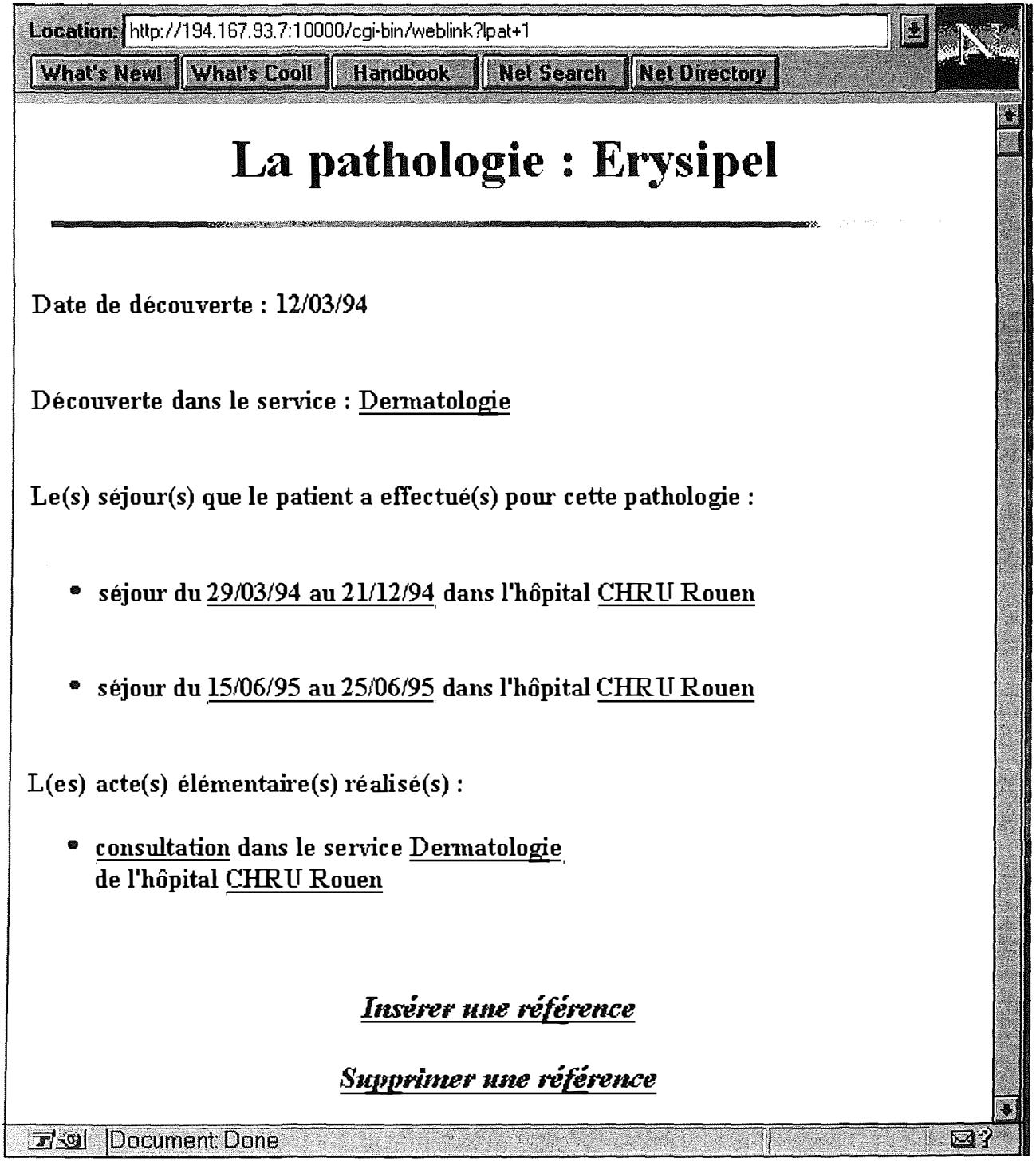


Figure 43 : Une pathologie du patient

Le médecin veut en savoir plus à propos de l'acte élémentaire 'consultation' qui a été réalisé auparavant (*Figure 44 : Les actes élémentaires*). Il constate qu'il y a une référence à cet acte.

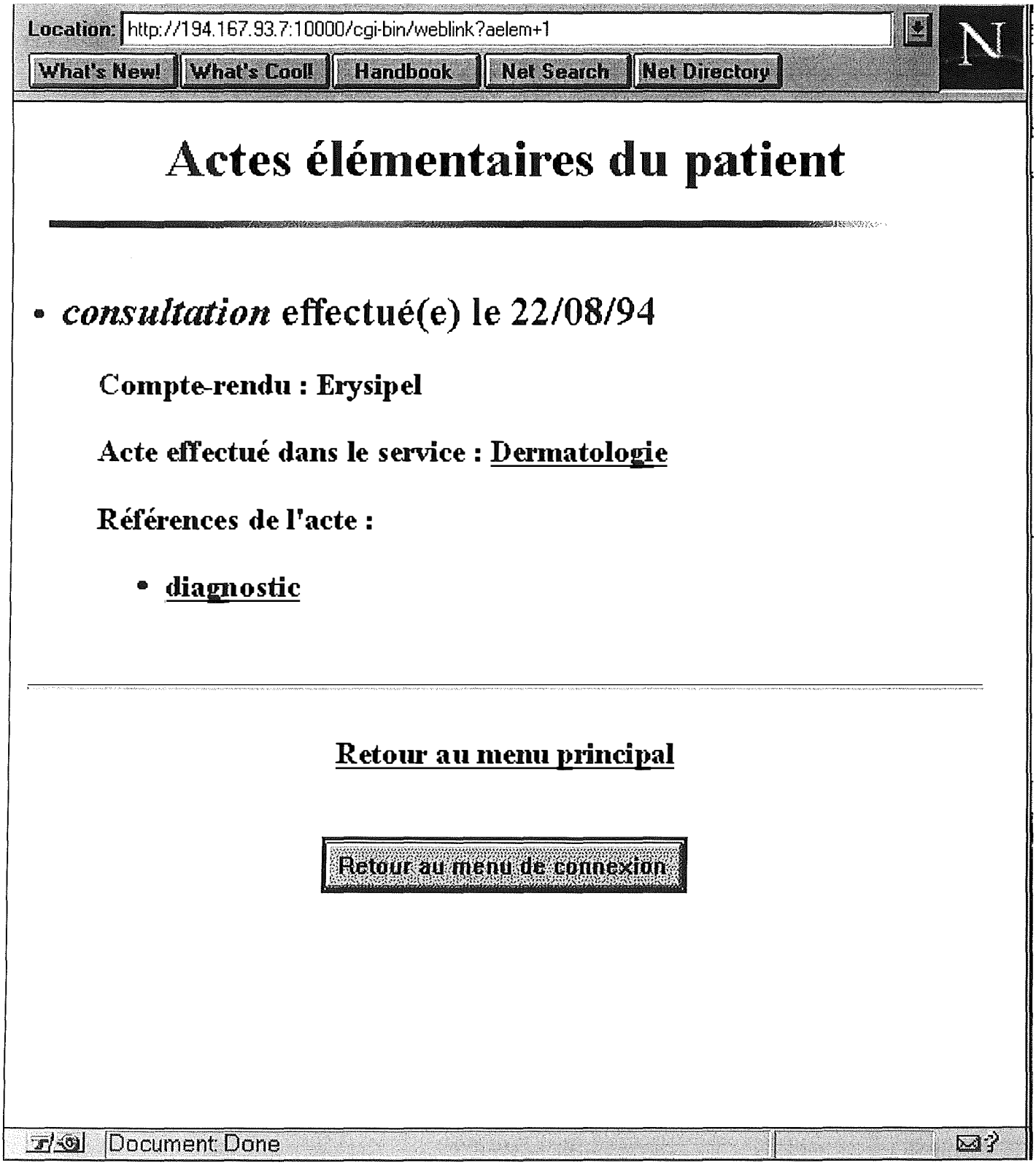


Figure 44 : Les actes élémentaires

Il accède à cette référence et obtient une photographie de la plaie du patient (*Figure 45 : Une référence d'un acte élémentaire*).

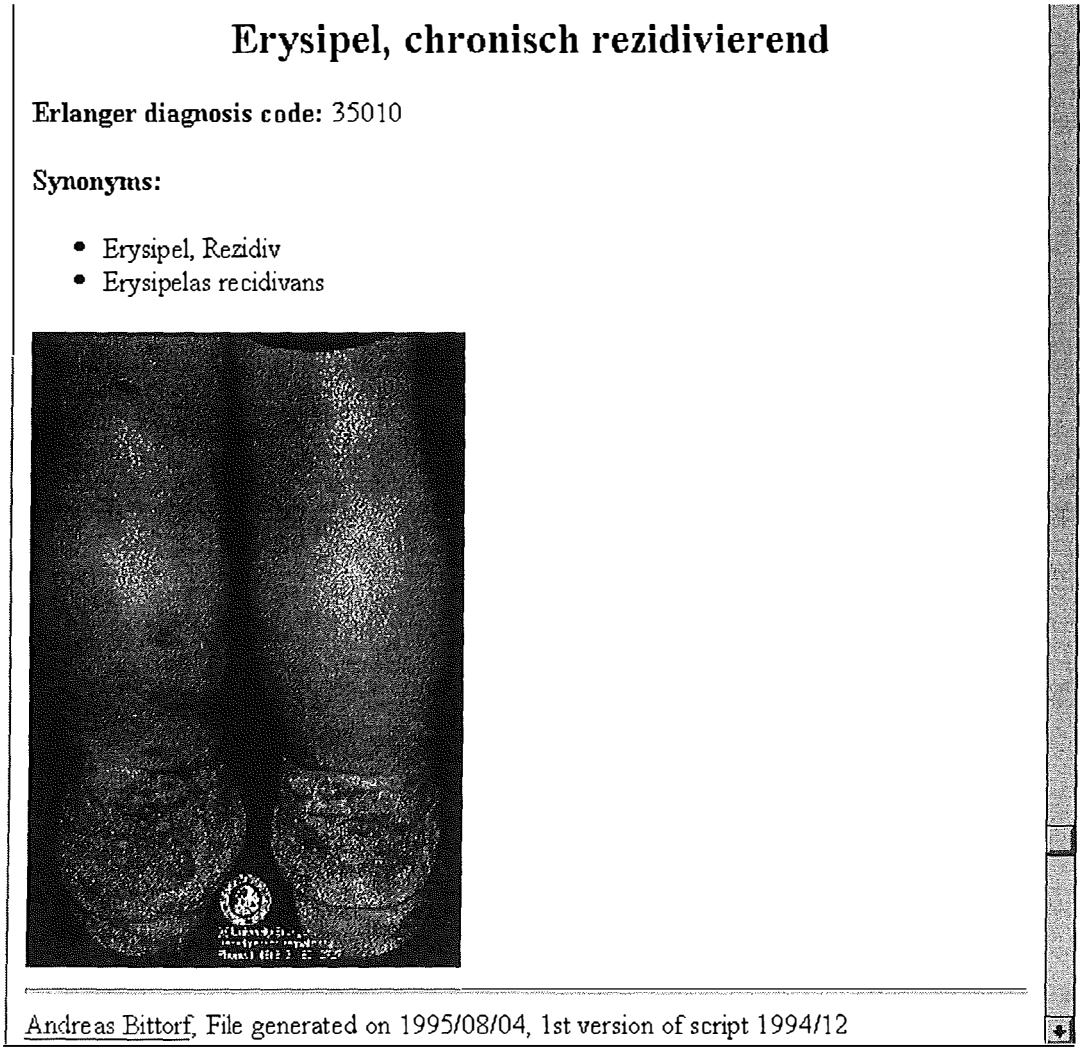


Figure 45 : Une référence d'un acte élémentaire

Dans chaque fenêtre, on peut retourner au menu qui correspond à notre fonction dans le monde hospitalier (dans notre cas, un retour vers le menu du médecin). Il est également possible de se rendre au menu de connexion. Dans ce cas, la carte est déconnectée.

La richesse de notre application se fait ressentir par l'intermédaire des références contenues dans la carte. Par exemple, la photographie de la plaie du patient ne peut pas dans l'état actuel de la technologie des cartes à microprocesseur (cfr point 2.3 du cahpitre II) être contenue dans la carte. Nous maintenons uniquement dans la carte la référence vers cette ressource.

6. Conclusions

Ce chapitre nous a permis de présenter notre application carte à microprocesseur. Nous avons voulu montrer au lecteur la façon dont nous avions travaillé pour la développer. Nous sommes partis d'un cahier des charges et d'un schéma entité-association pour en arriver à un code source et à des fenêtres.

Nous avons appliqué autant que possible la méthode par prototypage transformationnel qui nous avait été enseignée au cours de Méthodologie de Développement de Logiciels. En effet, à plusieurs moments dans les cycles de développement, nous avons marqué un temps d'arrêt afin d'évaluer le produit avec le client.

Dans ce chapitre, nous avons voulu uniquement **montrer la partie consultation** des données contenues dans la carte CQL. Nous ne tenions pas à alourdir l'exposé avec les étapes d'insertions et de modifications de données. Vous pourrez constater lors de la démonstration que dans la pratique **ces fonctionnalités ont été développées**.

Chapitre V :
Les perspectives

1. Introduction

Lors de ce dernier chapitre, nous nous efforcerons de présenter les perspectives qui découlent de notre travail au laboratoire RD2P et du développement de notre application. La première section concerne la sécurité dans le monde de l'Internet. Il reste certainement encore beaucoup de choses à faire dans ce domaine, mais nous proposerons déjà un aperçu de l'utilisation de la cryptographie sous Internet.

La deuxième section de ce chapitre traite d'un prototype que nous avons réalisé dans le but de démontrer qu'il était possible de remplacer la plate-forme Corba par un composant plus léger pour le client. Nous avons utilisé à cet effet les sockets Unix.

2. La sécurité

2.1. Introduction

Dans cette première partie, nous avons l'intention de montrer qu'il est possible d'apporter des améliorations notables à notre application au point de vue de la sécurité. A long terme, l'objectif d'une telle application est de pouvoir **garantir la sécurité de bout en bout** (End-to-End Security). Cette notion laisse entendre une sécurité garantie tout au long de l'exécution de l'application. Jusqu'à présent, nous avions considéré que tout ce qui tourne autour de la sécurité ne nous intéressait pas. Il est toutefois évident qu'une telle application n'est acceptable que dans la mesure où elle est tout à fait sécurisée. Ni les médecins ni les patients n'accepteraient de voir circuler des informations aussi sensibles que des informations médicales sans que celles-ci ne soient protégées.

A l'heure actuelle, la communication sous Internet est un élément que les gens considèrent avec beaucoup de méfiance. Ainsi, nous allons vous présenter un des moyens disponibles pour **sécuriser les transactions dans le monde de l'Internet**. Il s'agit du protocole de transfert **S-HTTP (Secure HyperText Transfer Protocol)**. Ce protocole, comme nous le verrons dans la suite, utilise les fonctions cryptographiques pour sécuriser les transactions. Avant cela, nous nous attarderons quelque peu aux principes sous-jacents à la cryptographie.

2.2. Les principes de base de la cryptographie

Avant d'entrer plus dans les détails du protocole S-HTTP et afin de bien fixer les idées, nous vous proposons une brève évocation des principes de base de la cryptographie [SECU95]. Le premier paragraphe expliquera ce qu'il faut entendre par cryptographie symétrique. Le second s'attachera à expliciter la cryptographie asymétrique.

a) La cryptographie symétrique

Le chiffrement symétrique est basé sur l'existence **d'une seule clé**. Cette clé est utilisée aussi bien pour le chiffrement que pour le déchiffrement. D'où la notion de symétrie. Cette clé est appelée clé privée ou partagée. En effet, pour que le texte chiffré soit invulnérable, il faut que cette clé reste connue uniquement des deux parties. On peut schématiser la situation par la *Figure 46*.

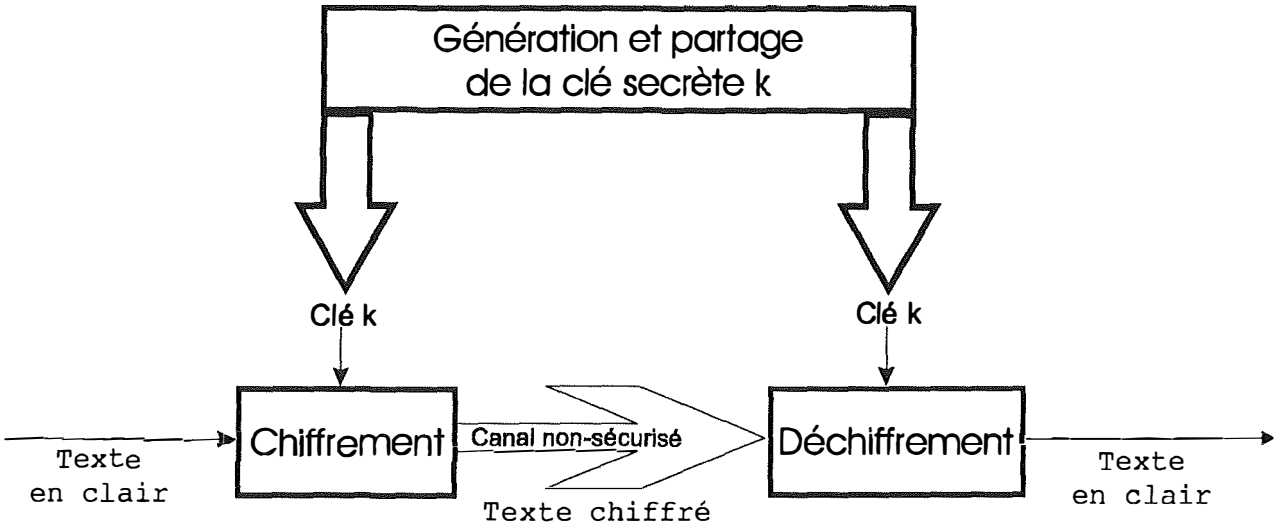


Figure 46 : La cryptographie symétrique

Les deux entités qui désirent s'échanger des informations de façon sécurisée ont un secret en commun : la clé k. Par contre, les algorithmes de chiffrement et de déchiffrement, notés respectivement E et D, peuvent être connus de tous.

Deux propriétés sont particulièrement intéressantes. La première nous indique que, même en connaissant le message chiffré et les algorithmes de chiffrement et de déchiffrement, on ne peut déduire aucune information à propos de la clé secrète ou du message de départ. Cela implique que tout changement, si minime soit-il, sur le message de départ modifie radicalement le message chiffré. Ce dernier apparaît donc comme un texte totalement aléatoire. La deuxième propriété est encore plus forte. En effet, si l'on connaît le message de départ, le message chiffré, les algorithmes de chiffrement et de déchiffrement, on ne peut déduire aucune information au sujet de la clé secrète.

Muni de ces deux propriétés, un algorithme de chiffrement n'est vulnérable que par l'essai exhaustif des clés. L'algorithme de chiffrement symétrique le plus connu est le DES (Data Encryption Standard). Il a été rendu public en 1974 et depuis lors il résiste toujours aux attaques des plus éminents chercheurs. Il est basé sur une clé secrète de 56 bits et il chiffre des blocs de texte de 64 bits. Il est caractérisé par sa rapidité.

b) Les fonctions à sens unique

Une fonction à sens unique se caractérise par le fait qu'il est facile de calculer $y = f(x)$ pour tout x appartenant au domaine de la fonction mais que le calcul de l'inverse est pratiquement infaisable. Du point de vue de la cryptographie, une fonction est dite à sens unique si elle est paramètrée par un élément secret.

Nous évoquons ces fonctions à sens unique pour deux raisons. Tout d'abord, le protocole S-HTTP utilise une de ces fonctions : MD5 (Message Digest 5). Ensuite, les algorithmes à clé publique sont en réalité un cas particulier de fonctions à sens unique. En effet, il s'agit de fonctions à sens unique pour lesquelles une "trappe" a été prévue. Si l'utilisateur connaît la trappe, le calcul de la fonction inverse devient un jeu d'enfant.

c) La cryptographie asymétrique

La cryptographie asymétrique est également appelée cryptographie à clé publique ou à clé révélée. Elle est basée sur un algorithme auquel sont associées **deux clés distinctes**. Le terme asymétrique provient du fait que l'émetteur utilise une clé et le récepteur une autre clé. La différence fondamentale par rapport au chiffrement symétrique est qu'il n'y a pas de secret partagé entre les deux interlocuteurs. La *Figure* 47 en schématise le fonctionnement.

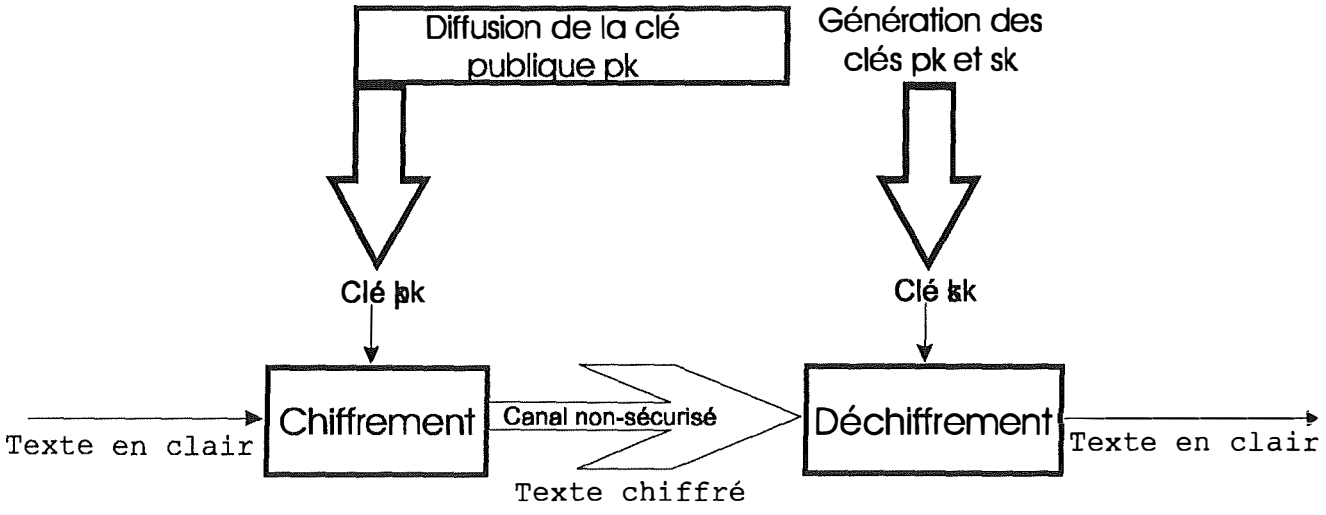


Figure 47 : La cryptographie asymétrique

Toute entité possède deux clés qui lui sont propres. L'une est la clé publique, connue de tous, et l'autre est la clé privée, connue uniquement de son propriétaire. Il existe une relation entre ces deux clés : tout message chiffré avec la clé publique peut être déchiffré avec la clé secrète. Tout comme dans le cas d'algorithmes à clé privée, les algorithmes de chiffrement et de déchiffrement sont connus de tous et notés respectivement E et D.

Les algorithmes à clé publique répondent aux deux propriétés que nous avons énoncées pour les algorithmes à clé secrète. En plus de cela, ils possèdent la propriété suivante : si l'on connaît la clé publique et les algorithmes de chiffrement et de déchiffrement, on ne peut déduire aucune information concernant la clé sécrète.

L'algorithme à clé publique le plus utilisé est sans conteste le RSA (du nom de ses trois auteurs : Rivest, Shamir et Adelman). Il a été publié en 1976. Il est basé sur la décomposition d'un nombre en facteurs premiers.

Envisageons au moyen d'un exemple les possibilités qu'offrent les algorithmes asymétriques. Alice et Bob désirent s'échanger des messages chiffrés. Ils possèdent chacun leur clé publique et leur clé privée ainsi que la clé publique de leur partenaire. Le premier exemple illustre le chiffrement pur et simple d'un message. Le second propose un système de signature du message.

Alice : p_{ka}, s_{ka} et p_{kb}
Bob : p_{kb}, s_{kb} et p_{ka}

p_{ka} : clé publique d'Alice
s_{ka} : clé secrète d'Alice
p_{kb} : clé publique de Bob
s_{kb} : clé secrète de Bob

1. Alice envoie un message chiffré M' à Bob. Le message M est chiffré avec l'algorithme E et la clé publique de Bob.
   $M' = E (M , p_{kb})$
   Bob reçoit le message M' de Alice. Le message M' est déchiffré avec l'algorithme D et la clé privée de Bob pour obtenir le message de départ M.
   $M = D (M', s_{kb})$

2. Bob craint que ce ne soit pas Alice qui lui écrit. Alice va donc signer son texte. Elle chiffre son document avec sa clé secrète.
   $M' = E (M , s_{ka})$
   Bob déchiffre le message avec la clé publique d'Alice. Tout le monde pourrait effectuer cette opération mais ce qui importe pour Bob c'est d'être sûr qu'il est en correspondance avec Alice.
   $M = D (M', p_{ka})$

2.3. S-HTTP

2.3.1. Introduction

Secure NCSA httpd est un serveur World Wide Web qui garantit la sécurité des transactions et leur authentification [SHT95] [ReSc94]. Il peut assurer ces fonctions quand il est en communication avec d'autres serveurs sur le WWW utilisant également le S-HTTP. Secure NCSA httpd a été développé par la société Enterprise Integration Technologies avec la collaboration du RSA Data Security et du NCSA (National Center for Supercomputing Applications).

Un besoin croissant de protéger des données à caractère sensible et des données qui ont une certaine valeur s'est fait ressentir dans le milieu de l'Internet; les données y circulant ne bénéficiaient de pratiquement aucune protection. C'est une des raisons pour laquelle, S-HTTP a été développé. De plus, les individus et les sociétés qui communiquent par la voie de l'Internet désirent également disposer de fonctions telles que l'identification et l'authentification d'un document ou d'un interlocuteur.

Le développement de la sécurité sur l'Internet devient un problème crucial. En particulier, les sociétés qui actuellement mettent en place des marchés électroniques ont besoin de ce type de garantie. Il est tout à fait nécessaire de développer des modes de fonctionnement de client à serveur et de serveur à client qui soient à la fois parfaitement sécurisés et peu contraignants pour les clients.

Envisageons par exemple la situation suivante :

Un client désire acheter un caméscope sur un catalogue on-line. Comment le client peut-il être sûr qu'il est devant le véritable catalogue de la société qu'il désire ? Comment le client sait-il si les informations qu'il envoie (numéro de carte de crédit, nom, adresse, ...) parviendront réellement au fournisseur approprié ?

En effet, il est possible qu'un tiers se soit inséré dans le réseau à l'insu de tout le monde. Il faudrait donc que le client puisse authentifier le catalogue qu'il a reçu d'un serveur WWW. De la sorte, il pourrait obtenir la preuve qu'il est en communication avec le véritable catalogue. Poursuivons notre exemple :

Un fois que le client a vérifié qu'il était en contact avec le fournisseur légitime, il peut vouloir acheter le caméscope. Comment le client pourra-t-il procurer, d'une façon sécurisée, au vendeur les informations utiles au règlement de l'achat ? Comment le magasin enverra-t-il la confirmation de commande au client ?

Nous sommes face à un nouveau problème. L'information envoyée du client au fournisseur ainsi que la confirmation de la commande doivent être protégées. Jusqu'à présent, nous nous sommes plutôt penchés du côté de la protection du client. L'exemple suivant montre qu'il est également important de protéger le fournisseur. D'ailleurs, si le fournisseur ne bénéficie pas d'un certain nombre de garanties, il quittera bien vite le marché électronique.

Si vous êtes le patron d'un grand journal électronique disponible sur le Web, comment pouvez-vous savoir quelles sont les personnes qui accèdent à votre serveur ?

Le fonctionnement des journaux électroniques est basé sur le système d'abonnement. Une personne qui désire accéder à un journal électronique quotidiennement paye un abonnement. Le problème pour le patron du journal est de pouvoir vérifier que la personne qui consulte son journal a effectivement payé son abonnement et qu'elle ne se fait pas passer pour une autre personne. Il s'agit donc de la problématique de l'authentification.

2.3.2. Les concepts de base

Cette partie a pour but d'introduire les notions utilisées dans S-HTTP. Nous allons construire un exemple de transactions commerciales possibles entre une société ConnectorLand et des clients. Nous allons montrer que ces transactions nécessitent des modes de sécurité différents suivant l'opération en cours.

a) Initialisation

Dans le but de créer un serveur WWW sécurisé, un responsable du serveur (webmaster) doit **définir d'une façon précise l'ensemble des transactions qu'un client pourra effectuer avec le serveur**. Une transaction est composée d'une demande et d'une réponse. Le responsable du serveur doit établir ce niveau de sécurité pour chacun des éléments de toutes les transactions. Par exemple, lors d'un accès à la home page de ConnectorLand, le webmaster décide que cette page, envoyée au client, sera signée. De cette manière, le client pourra vérifier qu'il est bien en communication avec la société ConnectorLand et non pas avec un tiers qui se ferait passer pour celle-ci. Il semblerait également vraisemblable que lors du paiement d'une commande le client signe et chiffre son document.

b) Signature digitale par le serveur

Le client sélectionne l'hyperlien vers la firme concernée. Cela correspond à la demande de la part du client pour obtenir la home page de chez ConnectorLand. Le serveur, qui reçoit la demande, envoie la home page signée. De telle sorte que le client puisse vérifier la provenance de la home page.

```
<head><!--#certs name="foo"--></head>
<body>
<title>Encryption and signature Demo</title>
<h1> Encryption and signature Demo </h1>
<form action="shttp://www.commerce.net:8002/cgi-
bin/examples/sign.html"
DN = '<!-- name="foo"-->
CRYPTOPTS="SHTTP-Privacy-Enhancements: recv-
required=sign,encrypt">
This is an encryption/signature demo. The document will be
signed using the PKCS-7 encapsulation format </p>
Please enter a secret here : <input size=30 name="secret">
</p>
To submit your request, please click :
<input type=submit name="GO">
</form></body>
```

Figure 48 : La signature par le serveur

La signature digitale fonctionne de la façon suivante. Le serveur applique une fonction à sens unique, appelée également fonction de hachage (message digest), au document HTML qu'il va envoyer au client. Il chiffre le résultat *r* de la fonction de hachage avec sa clé privée. Il envoie le tout au client. Celui-ci déchiffre r au moyen de la clé publique du serveur, effectue la fonction de hachage sur le document HTML et obtient le résultat *r'*. Si $r = r'$, le document fourni au client est le document officiel de chez ConnectorLand.

Suite à ce premier exemple décrit à la *Figure 48*, nous avons déjà constaté quelques particularités. Tout d'abord, les hyperliens qui référencent un site sécurisé sont d'un type un peu particulier. La méthode de transport n'est plus comme à l'accoutumée *http://* mais elle devient *shttp://*. Ensuite, tous les hyperliens sécurisés contiennent un nom distinct (Distinguished Name). Celui-ci a pour but d'identifier d'une façon unique la clé publique du serveur que le client utilisera pour chiffrer sa demande.

`DN = <!--#dn name="foo"-->` Cette instruction indique au serveur qu'il doit insérer à cet endroit le distinguished name (DN) complet. La syntaxe du nom distinct correspond au standard X.500. Par exemple, l'instruction ci-dessus sera remplacée par `DN ="CN=shttpd-sample, OU=Persona Certificate, O=RSA Data Security, Ins., C=US"`. CN représente le nom commun (Common Name), OU l'unité organisationnelle (Organisational Unit), O l'organisation (Organization) et C le pays (Country).

Enfin, le certificat de la clé publique est inséré dans le document au moyen de l'instruction suivante : `<!--#certs name="foo"-->.`

c) Le chiffrement sans signature digitale

A partir de la home page de ConnectorLand, le client peut naviguer à travers le serveur de cette société. Par exemple, il pourrait décider de consulter le catalogue de la société. S'il désire passer une commande, le serveur lui propose un bulletin de commande. Si l'on examine le contenu de celui-ci, on constate la présence d'un nouvel attribut : CRYPTOPTS (les options cryptographiques).

CRYPTOPTS="SHTTP-Privacy-Domains: orig-required=PKCS-7; recv-required = PKCS-7; SHTTP-Key-Exchange-Algorithms: orig-optional = RSA; recv-required = RSA;

SHTTP-Signature-Algorithms: orig-required = RSA, recv-required = RSA;
SHTTP-Message-Digest-Algorithms: orig-required = MD5; recv-required = MD5;
SHTTP-Privacy-Enhancements:orig-optional = signature, encrypt; recv-required = encrypt;"

Les options cryptographiques spécifient les éléments suivants : les algorithmes utilisés pour les échanges de clés, pour la signature et pour le hachage. La dernière ligne indique les exigences du CGI en matière de chiffrement. D'une façon globale, *orig* correspond aux conditions que doit remplir le client et *rec* celles du serveur.

On peut avoir trois types de conditions : required, optional ou refused. La première signifie que la personne concernée doit absolument remplir cette condition. Dans notre exemple ci-dessus, l'algorithme de signature utilisé par le client doit être le RSA. La condition optional signifie qu'elle n'est pas obligatoire. L'algorithme d'échange de clés pour le client pourrait être le RSA. Enfin, la condition refused spécifie les options qui ne peuvent être utilisées par un interlocuteur. Par exemple, la condition *recv-refused = encrypt* indique que le serveur refuse de recevoir un message chiffré d'un client.

Expliquons maintenant le cas précis de notre exemple. Le client doit chiffrer sa commande de matériel à la société (cfr. *SHTTP-Privacy-Enhancements: recv-required = encrypt)*. Pour cela, il utilise le système de la clé de transaction. Une clé symétrique (ou clé de transaction) est générée aléatoirement par le client. Il chiffre la demande avec cette clé. La clé elle-même est chiffrée avec la clé publique du serveur. Enfin, la clé de transaction (chiffrée) et la demande (chiffrée également) sont envoyées au serveur.

Quand il reçoit l'ensemble, le serveur déchiffre la clé de transaction avec sa clé secrète. De telle sorte qu'il puisse déchiffrer le message avec la clé de transaction. On comprend bien les motivations de la société à exiger que les commandes lui parviennent chiffrées. Tout d'abord, pour protéger la vie privée de ses clients. Mais également pour des raisons économiques. En effet, la société ConnectorLand ne désire pas que ses concurrents soient au courant du contenu des commandes de ses clients.

d) La signature digitale par le client

La société ConnectorLand veut restreindre l'accès de certaines parties de son catalogue à un certain nombre clients privilégiés. Ces personnes qui effectuent des achats fréquents appartiennent à un Gold Club.

Afin de pouvoir reconnaître les clients qui appartiennent à ce Gold Club, la société exige que le client signe sa demande lors de tout accès aux services du Club. Ce mécanisme se traduit par les options cryptographiques suivantes :

CRYPTOPTS="SHTTP-Privacy-Domains: orig-required=PKCS-7; recv-required = PKCS-7;
SHTTP-Key-Exchange-Algorithms: orig-optional = RSA; recv-required = RSA;
SHTTP-Signature-Algorithms: orig-required = RSA, recv-required = RSA;
SHTTP-Message-Digest-Algorithms: orig-required = MD5; recv-required = MD5;
SHTTP-Privacy-Enhancements:orig-optional = signature, encrypt; recv-required = encrypt, **sign**;"

Quand l'utilisateur désire bénéficier d'un des privilèges du Gold Club, il signe sa demande. Pour ce faire, il applique l'algorithme de hachage (dans notre exemple, le MD5) à sa demande. Le résultat est chiffré en utilisant sa clé privée. Ensuite, le résultat chiffré, le certificat de la clé publique du client et la demande sont envoyés au serveur.

De son côté, le serveur utilise la clé publique du client pour déchiffrer le résultat. Il applique l'algorithme de hachage à la demande et vérifie les deux résultats. S'ils sont identiques, l'intégrité du document est garantie.

e) Les échanges de clés

Les algorithmes à clé symétrique présentent un avantage important : ils sont plus rapides que les algorithmes à clé publique. Par contre, ils présentent le désavantage d'engendrer un accord préalable sur un secret partagé entre les deux interlocuteurs.

Le protocole S-HTTP propose une solution qui est d'ailleurs couramment utilisée dans le monde de la cryptographie. Il suggère d'appliquer un algorithme à clé privée (symétrique) sur le message et d'ensuite protéger la clé privée par l'application du RSA avec la clé publique du destinataire, de telle manière que le destinataire soit le seul à pouvoir déchiffrer la clé de transaction et donc à retrouver le message. Dans sa réponse au client, le serveur utilisera soit la clé privée soit il génèrera une nouvelle clé privée par le même mécanisme.

En utilisant ce système, on évite **la difficulté du partage initial du secret et on gagne un certain temps de calcul** grâce à l'utilisation d'un algorithme à clé privée sur la plus grande partie du message; l'algorithme à clé publique n'est en effet utilisé que sur le très court message que représente la clé privée.

f) L'authentification et le mot de passe

Une autre technique basée sur un secret partagé peut être utilisée. Ce secret prend la forme par exemple de couples de noms d'utilisateur et de mot de passe. Ces couples sont chargés dans une base de données du client et du serveur.

L'option cryptographique correspondante est la suivante : *SHTTP-Privacy-Enhancements:orig-optional = signature, encrypt; recv-required =* ***auth***. Quand l'utilisateur sélectionne l'hyperlien en question, il est invité à ne pas chiffrer sa demande. En effet, le système examine la base de données comprenant les couples d'authentification. Si l'utilisateur ne possède qu'un seul couple de données partagées, celui-ci est envoyé au serveur en même temps que la demande. Si l'utilisateur possède plusieurs clés, le client lui demandera de choisir l'une d'elle.

Pour sa part, le serveur, quand il reçoit les informations décrites ci-dessus, vérifie que le couple provenant du client correspond bien à un couple présent dans sa base de données. Si la vérification prouve l'identité du demandeur, le serveur fournira à l'utilisateur les documents demandés.

g) L'échange extérieur de clés

Une autre méthode est basée sur la même logique. Le client et le serveur partagent un secret sous la forme d'un couple composé d'un nom d'utilisateur et d'un mot de passe. A chaque couple correspond une clé secrète. Ces trois éléments sont placés dans une base de données. Le but de cette procédure est de permettre l'authentification et le chiffrement avec des clés secrètes.

L'option cryptographique qui autorise ces fonctionnalités est la suivante : *SHTTP-Privacy-Enhancements:orig-optional = signature, encrypt; recv-required =* ***auth, encrypt***. Lorsqu'il sélectionne l'URL correspondant, le client génère une clé de transaction. Cette clé est utilisée pour chiffrer le message. Elle est elle-même chiffrée par la clé secrète. Le client envoie au serveur : le message chiffré, la clé chiffrée, le nom d'utilisateur et le mot de passe.

A la réception, le serveur vérifie l'identité de la personne qui lui a envoyé le message. Si le nom de l'utilisateur et le mot de passe sont présents dans la base de données, le processus de déchiffrement est entamé.

2.3.3. S-HTTP et les Common Gateway Interface (CGI)

S-HTTP fournit un ensemble de variables d'environnement. Celles-ci permettent au programme **CGI d'obtenir des informations importantes** concernant les moyens cryptographiques qui ont été utilisés lors d'une transaction. Comme nous avons déjà pu le constater, ces variables peuvent être employées pour choisir le type de réponse qui sera fournie au client.

Détaillons quelque peu ces variables :

SHTTP_PROCESS : cette variable indique au CGI le type de cryptographie qui a été utilisé dans la demande qui lui parvient. Quatre valeurs sont possibles : SIGNED, ENCRYPTED, AUTHENTICATED, NONE. Une combinaison de ces éléments est bien

évidemment possible. Dans ce cas, les éléments sont séparés par '_AND_'. Par exemple, un message qui aurait été signé et chiffré aurait la valeur suivante pour cette variable : SIGNED_AND_ENCRYPTED.

SHTTP_AUTH_USER : cette variable d'environnement fournit le nom de l'utilisateur qui a envoyé la demande. Elle est utile dans le cas de l'authentification, comme nous l'avons présenté dans les deux paragraphes précédents. Elle ne possédera une valeur que si la variable SHTTP_PROCESS comporte la valeur AUTHENTICATED.

SHTTP_SIGNER : c'est une variable qui représente le nom distinct de la personne qui a appliqué une signature à un document.

SHTTP_SIGNER_CERT_CHAIN : cet élément contient le certificat du signataire.

SHTTP_SIGNER_CHAIN_LEN : elle spécifie la longueur du certificat.

SHTTP_VERSION : cet élément indique la version du S-HTTP qui est utilisée par le client.

SHTTP_HEADER_DATA : cette variable contient les options cryptographiques que le client a appliquées à son document.

2.3.4. La modification des réponses du serveur

Un serveur S-HTTP peut **moduler le niveau de cryptographie qu'il utilisera pour répondre à ses clients** : chiffrer, signer, chiffrer et signer ou bien ni chiffrer ni signer. Deux possibilités s'offrent au gestionnaire d'un serveur. Soit il détermine le niveau cryptographique au moyen d'un fichier "dot". Soit il utilise les entêtes des CGI.

Dans le premier cas, à chaque fichier on fait correspondre un fichier "dot". Par exemple, au fichier "intro.html" correspondra un fichier ".intro.html". Ce fichier est appelé un fichier de configuration de sécurité locale. Le fichier de départ peut aussi bien être un fichier statique (ne contenant aucun traitement dynamique) qu'un fichier CGI. On retrouve plusieurs types d'instructions dans ces fichiers.

Par exemple, une instruction que nous avons déjà rencontrée auparavant sous une autre forme : *SHTTPPrivacyEnhancements sign, encrypt.* Elle indique au serveur que quand le fichier en question sera exécuté, le résultat de l'exécution doit être signé et chiffré. On peut combiner les valeurs suivantes : *sign, encrypt et auth.* D'autres instructions sont plutôt de l'ordre du contrôle d'accès. L'instruction *<link GET> require shttp </link>* refuse l'accès au fichier si la demande d'accès ne provient pas d'un client utilisant le protocole S-HTTP. Une instruction *SHTTPAuthAcceptencrypt* signifie que le serveur n'acceptera que les documents qui lui parviennent chiffrés. Dans celle-ci, *encrypt* peut être remplacé par *sign ou auth.* Si l'on

désire combiner plusieurs caractéristiques de cette dernière instruction, on insère deux lignes distinctes dans le fichier.

Le second cas concerne les options cryptographiques que l'on peut insérer dans l'entête d'un CGI. *SHTTP-Privacy-Enhancements:orig-optional = signature, encrypt; recv-required = auth.* Cette instruction contraint le serveur à répondre d'une façon chiffrée et signée et le client à s'authentifier. Bien sûr, une combinaison de ces différentes valeurs est toujours possible.

Les deux types de protection pourraient être combinés. On aurait en même temps un fichier "dot" et une entête dans le CGI. Un conflit à propos des protections pourrait survenir. Pour cette raison, un ordre de priorité a été défini. Les entêtes de CGI sont prioritaires par rapport au fichier "dot".

2.4. Secure NCSA Mosaic

2.4.1. Introduction

Secure NCSA Mosaic est une implémentation du protocole S-HTTP que nous avons explicité dans la section précédente [SMos95]]. Nous allons présenter très concrètement la façon dont les fonctionnalités du protocole S-HTTP ont été introduites dans un browser standard. Quand on analyse les différentes possibilités offertes par S-HTTP, on pourrait penser que cela complique fortement le travail de l'utilisateur. Nous allons montrer qu'il n'en est rien et que la cryptographie ne risque pas de détourner l'attention des utilisateurs.

2.4.2. L'application des concepts de base

a) Les éléments d'interface

Quand vous recevez un document HTML, Secure NCSA Mosaic vous indique, grâce à **un icône, quel niveau de sécurité a été appliqué à ce document**. La *Figure 49* présente les quatre icônes disponibles. Le premier représente un document qui n'a été ni signé ni chiffré. L'enveloppe signifie que le document est chiffré. Le sceau sur le document indique qu'il est signé. Enfin, le sceau sur l'enveloppe représente un document à la fois signé et chiffré.

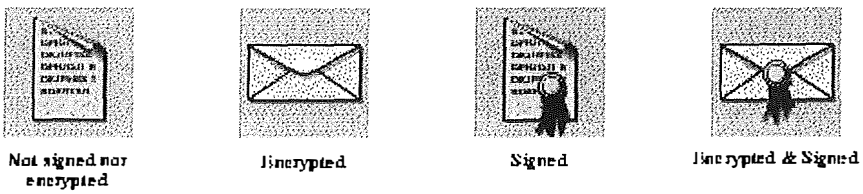


Figure 49 : Les icônes du niveau de sécurité

Lorsque Secure NCSA Mosaic applique une des fonctions cryptographiques, dont il dispose soit au document qu'il reçoit, soit à celui qu'il envoie, **un icône spécifique illustre l'activité qui se déroule**. Dans la *Figure 50*, nous constatons que le premier icône représente une activité de chiffrement ou de déchiffrement. On retrouve ensuite respectivement les activités de signature et de chiffrement, de signature, de vérification de la signature et de déchiffrement et enfin de vérification de la signature.

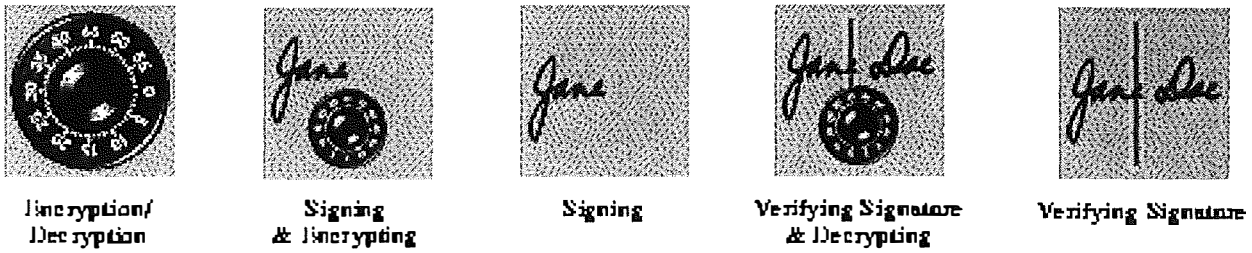


Figure 50 : Les icônes de traitement

Avant de détailler davantage les fonctions offertes par Secure NCSA Mosaic, nous précisons un élément important. On ne peut pas accéder à un hyperlien S-HTTP via la boîte de dialogue "open URL". L'accès doit obligatoirement avoir lieu à travers un document HTML. S'il n'en va pas de la sorte, une erreur sera signifiée à l'utilisateur.

b) La génération de signature chez le client

Secure NCSA Mosaic fournit à l'utilisateur la possibilité de signer ses documents. Ainsi, l'utilisateur sélectionne un lien. Si ce lien s'est vu attribuer par le serveur une propriété qui nécessite la signature du demandeur, une boîte de dialogue *signature* est ouverte. Dans celle-ci, l'utilisateur peut choisir une de ses clés publiques. Il peut également consulter le certificat d'une de ses clés. Si le bouton *sign* est sélectionné, Secure NCSA Mosaic ajoute une signature à la demande de document et envoie l'ensemble au serveur concerné.

c) La génération de clés publiques et de clés privées

Comme nous avons déjà pu le constater, le protocole S-HTTP est basé en partie sur l'algorithme RSA à clé publique. Pour mettre en place cet algorithme, chaque entité doit posséder une paire de clés. Secure NCSA Mosaic permet la gestion et la conservation de ces paires dans une base de données.

Secure NCSA Mosaic génère une paire de clés : l'une publique et l'autre privée. Ensuite, il effectue une demande auprès d'une agence de certification pour obtenir un certificat pour sa paire de clés. Si la demande et les documents accompagnant celle-ci répondent aux exigences de l'organisme de certification, l'organisme fournit à Secure NCSA Mosaic un certificat de clé publique. Ce certificat est inséré dans la base de données qui contenait déjà les paires de clés.

Cette base de données est protégée par un mot de passe. En effet, la première fois qu'un utilisateur demande la génération d'un paire de clés, il doit fournir un mot de passe. Celui-ci est utilisé pour chiffrer les clés qui sont contenues dans la base de données. A chaque première sélection d'un lien S-HTTP qui nécessite l'utilisation d'une paire de clés, ce mot de passe est exigé. Une fois que celui-ci a été correctement introduit, on n'exigera sa présentation que lors de la prochaine exécution de Secure NCSA Mosaic.

Cela signifie que si l'utilisateur laisse son ordinateur sans surveillance, après avoir présenté son mot de passe, il court le risque qu'un utilisateur malveillant utilise les clés de l'autre utilisateur. Lorsqu'il s'en va, l'utilisateur doit fermer Secure NCSA Mosaic.

Quand l'utilisateur demande la génération d'une paire de clés, il se voit présenter un formulaire contenant différents éléments. Le premier concerne la longueur de la clé. Le second spécifie la période de validité de la clé. En effet, la validité d'une clé est comprise entre 1 et 52 semaines. Ensuite, l'utilisateur doit insérer son nom distinct qui permettra de l'identifier parmi les individus, les organisations et les serveurs. Il est à noter que suivant l'organisme de certification auquel on fait appel, certains champs sont facultatifs.

Une fois que l'on a inséré les renseignements exigés ci-dessus, Secure NCSA Mosaic vous demande d'indiquer l'endroit où vous allez stocker la demande de certificat de clé publique. Cela étant fait, Secure NCSA Mosaic génère la paire de clés et la demande de certificat. L'avant-dernière étape consiste à envoyer cette demande à une agence de certification. Si vous générez votre première paire de clés, Secure NCSA Mosaic vous demandera d'introduire une séquence aléatoire qui se matérialise par des déplacements de la souris.

Quand vous recevez le certificat de clé publique de votre agence de certification, il vous reste à l'insérer dans Secure NCSA Mosaic. Pour ce faire, vous sélectionnez l'élément "Add Certificate" du menu "Key" et vous précisez où se situe le certificat. Le système est maintenant prêt pour gérer des documents cryptographiques.

2.4.3. Le changement de mot de passe

Comme nous l'avons expliqué ci-dessus, le mot de passe sert à chiffrer la base de données contenant les clés. A un moment donné, on peut désirer changer ce mot de passe. Il suffit de choisir l'élément "Set Password" dans le menu "Key". Il faudra alors entrer l'ancien mot de passe et deux fois le nouveau. A partir de cet instant, la base de données est protégée avec le nouveau mot de passe.

2.4.4. La vérification de l'origine d'un document

Secure NCSA Mosaic offre la possibilité d'examiner les propriétés d'un point de vue de la sécurité d'un document. En effet, quand on reçoit un document important, on désire être sûr de sa provenance. Comme nous l'avions montré ci-dessus, à chaque document correspond un icône décrivant le niveau de sécurité. En cliquant sur cette icône, une fenêtre appelée

"Document Security Status Window" apparaît à l'écran. Elle fournit des informations telles que l'algorithme de chiffrement, de signature ou d'authentification qui a été utilisé.

Dans le cas, par exemple, de l'utilisation d'une clé publique, un premier certificat de clé publique atteste l'identité du signataire. Un second et les suivants se portent garants des certificats de clés publiques précédents. Une croix rouge en travers d'un certificat indique que celui-ci n'est pas valable.

2.4.5. La vérification du niveau de sécurité d'un hyperlien

Avant de sélectionner un hyperlien, l'utilisateur peut examiner les options cryptographiques que celui-ci contient. Il lui suffit de cliquer avec le bouton droit de la souris sur le lien. Une fenêtre nommée "Anchor Security Properties" est visible à l'écran. Cette fenêtre, identique à celle que nous avons décrite dans le paragraphe précédent, nous permettra par exemple de consulter les types d'algorithmes utilisés et le nom des entités qui sont référencées par l'hyperlien.

2.4.6. La clé "root"

La clé "root" a une importance cruciale dans le système. En effet, c'est la clé publique associée à l'agence de certification. C'est donc elle qui est utilisée dans toutes les communications avec cette agence. Or, pour que Secure NCSA Mosaic puisse déterminer la validité d'un certificat, il doit pouvoir en vérifier l'authenticité auprès de l'agence de certification.

2.4.7. L'authentification par la clé partagée

Ce dernier mécanisme permet à l'utilisateur de s'authentifier par l'intermédiaire d'un secret partagé. En l'occurrence, il s'agit d'un nom d'utilisateur et d'un mot de passe. Ce schéma, quoique ressemblant à la pratique du protocole HTTP, est bien plus sûr que celui-ci.

Un accord doit exister entre le client et le serveur à propos de ce secret. L'utilisateur peut ensuite insérer son secret dans Secure NCSA Mosaic au moyen de la commande "New Shared Secret". Une fois ce secret inséré, il sera conservé dans la base de données qui contient également les paires de clé. Quand une authentification par clé partagée est demandée à l'utilisateur, une fenêtre s'affichera lui demandant de choisir une clé secrète.

2.5. Conclusions

Dans cette première partie, nous avons montré qu'il est tout à fait possible de sécuriser les transactions sur Internet. Nous ne prétendons pas que ces mécanismes sont d'une protection totale mais elles apportent toutefois déjà un certain nombre de garanties. Il serait intéressant d'obtenir davantage de renseignements sur la protection de la base de données locale qui contient la plupart des données sensibles.

Actuellement, la tendance dans le monde de l'Internet est en tout cas à l'utilisation de plus en plus fréquente de la cryptographie. Il y a en tout cas un problème que nous n'avons pas soulevé mais qui est bien existant; c'est celui du droit. Comment rendre compatibles des transactions sécurisées avec, par exemple, le droit français ? En effet, celui-ci proscrit toute utilisation de la cryptographie...

3. La carte chez le client

3.1. Introduction

Dans un premier temps, nous avions développé une architecture client/serveur au moyen de la plate-forme Orbix. Lors d'une présentation de notre logiciel auprès des responsables de Gemplus, ceux-ci ont émis le souhait de remplacer Orbix par une autre architecture. En effet, ils désirent utiliser l'application WebSanté comme une vitrine de leur produit CQL. Pour que cela soit intéressant, il faut que chaque client puisse consulter sa carte santé à partir de chez lui, sans pour cela disposer d'Orbix. Il était donc nécessaire de montrer la faisabilité de ce projet sans Orbix. Nous avons dès lors réalisé un prototype fonctionnant avec les sockets Unix.

Dans les paragraphes suivants, nous ferons dans un premier temps une petite présentation du Transport Level Interface (TLI) qui est le nouveau standard Unix pour la gestion des sockets. La deuxième partie présentera plus précisément ce que nous avons réalisé comme architecture.

3.2. Présentation des TLI

L'interface TLI se situe au même niveau que les sockets [SunOS40]. Elle fournit **un support au transfert de données entre deux utilisateurs**. Elle se veut indépendante de tout protocole ou toute famille de protocoles de la couche transport. Ainsi, les TLI peuvent aussi bien fonctionner au-dessus de la famille TCP/IP mais aussi au-dessus de toute autre famille.

Un utilisateur des TLI peut demander à utiliser les services d'un fournisseur de transport (transport provider) d'un type quelconque : tcp, udp, ... L'utilisateur se voit alors attribuer un point de transport (transport endpoint ou tep). Par l'intermédiaire de ce point de transport, l'utilisateur a accès à un certain nombre de requêtes destinées au fournisseur de transport. De plus, des événements provenant du fournisseur sont signalés à l'utilisateur au moyen de ce tep. Le fournisseur de transport est donc l'entité qui, d'une part met à la disposition de l'utilisateur des services de l'Interface de Transport et d'autre part signale à l'utilisateur un certain nombre d'événements.

3.2.1. Les fournisseurs de transport

Un fournisseur de transport particulier se caractérise par le style de communication qu'il autorise. Le *Tableau 7* présente les principaux types de fournisseurs que l'on retrouve habituellement :

TYPE :	DESCRIPTION :
T_CLTS :	communication sans connexion (du type UDP)
T_COTS :	communication avec connexion et possibilité de déconnexion brutale (du type TCP)
T_COTS_ORD :	communication avec connexion et déconnexion sans perte de message

Tableau 7 : Les styles de communication

3.2.2. Le service en mode connecté

Nous allons développer dans les paragraphes suivants les différentes étapes nécessaires à la gestion d'un service de transport en mode connecté. Ce service est caractérisé par quatre phases : la gestion locale, l'établissement de la connexion, le transfert de données et la fermeture de la connexion.

a) La gestion locale

La phase de gestion locale, illustrée par la *Figure 51*, définit les opérations entre le fournisseur de transport et l'utilisateur. Celui-ci peut, par exemple, **vouloir établir un canal de communication avec un fournisseur de transport**. Or, chaque canal est un point de transport unique. La fonction `t_open` permet à l'utilisateur de choisir le type de fournisseur qu'il désire et de fixer le point de transport.

Un utilisateur peut posséder plusieurs connexions avec un même fournisseur de transport. Pour les différencier, la fonction `t_bind` attribue une adresse à chaque canal.

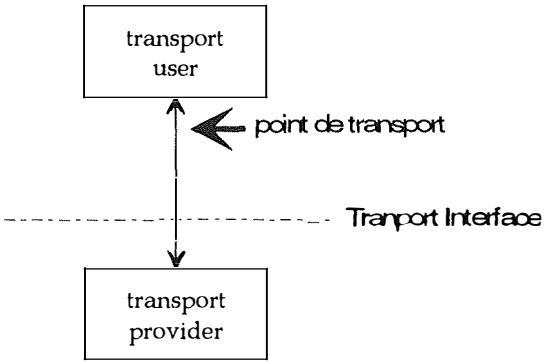


Figure 51 : La gestion locale

b) L'établissement de la connexion

La phase d'établissement de la connexion permet à deux utilisateurs de **créer une connexion entre eux**. Nous pouvons illustrer cela (cfr. *Figure 52*) par un modèle client/serveur. Le serveur s'est identifié auprès du fournisseur de transport et attend les éventuels clients; pour indiquer au fournisseur de transport que dorénavant il est prêt à recevoir les demandes de connexion, il utilise la fonction `t_listen`.

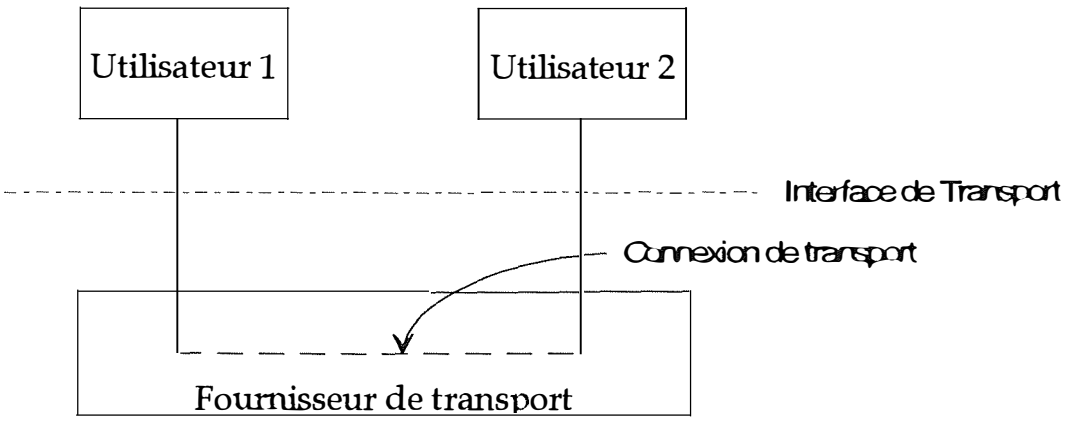


Figure 52 : L'établissement d'une connexion

Un client, désireux d'utiliser les services proposés par le serveur, se fait connaître du fournisseur de service (comme nous l'avons expliqué dans le point a). Le client utilise alors la fonction `t_connect` pour établir la connexion avec le serveur. L'adresse de ce dernier est un des paramètres de cette fonction.

Le serveur est averti qu'une demande de connexion vient de lui parvenir. La fonction `t_accept` signifiera au client que le serveur accepte sa demande. Dès la réception du `t_accept` par le client, la connexion est établie et les transferts de données peuvent commencer.

c) Le transfert de données

La phase de transfert permet aux utilisateurs de **transférer des données dans les deux sens**. La fonction `t_snd` permet l'envoi de données et la fonction `t_rcv` la réception. Cette dernière fonction est bloquante; cela signifie que les instructions suivantes ne seront pas exécutées tant que les données n'ont pas été reçues. Le service en mode connecté garantit que toute donnée envoyée par un utilisateur sera délivrée à l'utilisateur désiré dans l'ordre de son envoi.

d) La fermeture de la connexion

Cette dernière phase fournit un mécanisme pour **rompre une connexion entre deux utilisateurs**. Quand vous décidez que la communication est terminée entre vous et votre interlocuteur, vous pouvez demander au fournisseur de transport de fermer la connexion. Deux types de fermeture de connexion sont possibles.

Le premier, le plus brutal, signifie à votre interlocuteur que la communication est rompue, quel que soit l'état dans lequel les deux utilisateurs se trouvent. Ce type de fermeture a pour conséquence que toute information en cours de transmission est perdue. Les deux fonctions assurant ce type de fermeture sont `t_snddis` et `t_rcvdis`.

A l'inverse, le deuxième type de fermeture garantit une fin de connexion plus propre. Les données en cours de transmission seront envoyées à leur destinataire avant la fermeture de la connexion. Les fonctions `t_sndrel` et `t_rcvrel` assurent ce mécanisme.

3.3. L'architecture proposée

Comme nous l'avons déjà expliqué, l'architecture que nous avons proposée a pour but de remplacer la plate-forme Orbix. Les deux figures ci-dessous nous permettent de comparer les deux architectures.

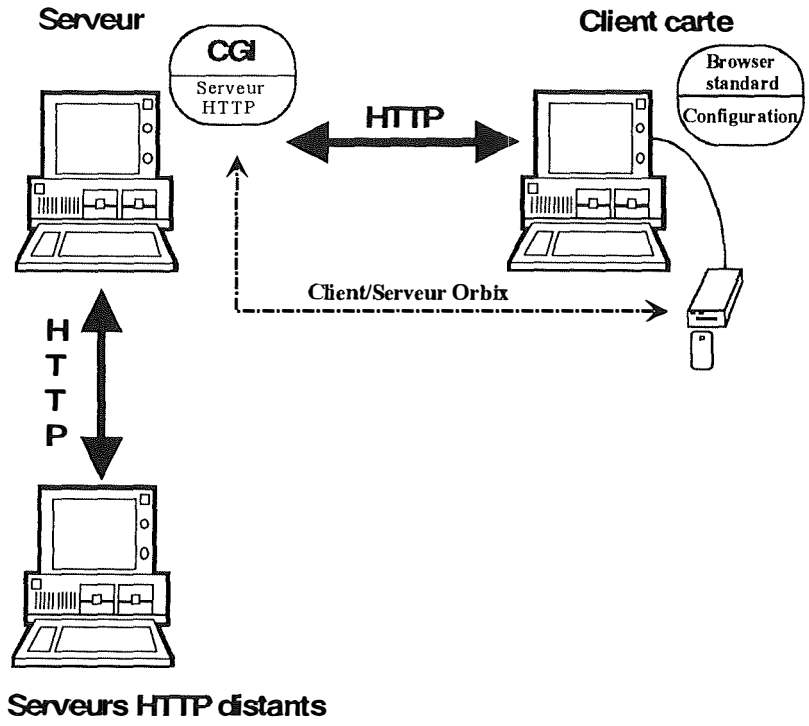


Figure 53 : L'architecture Orbix

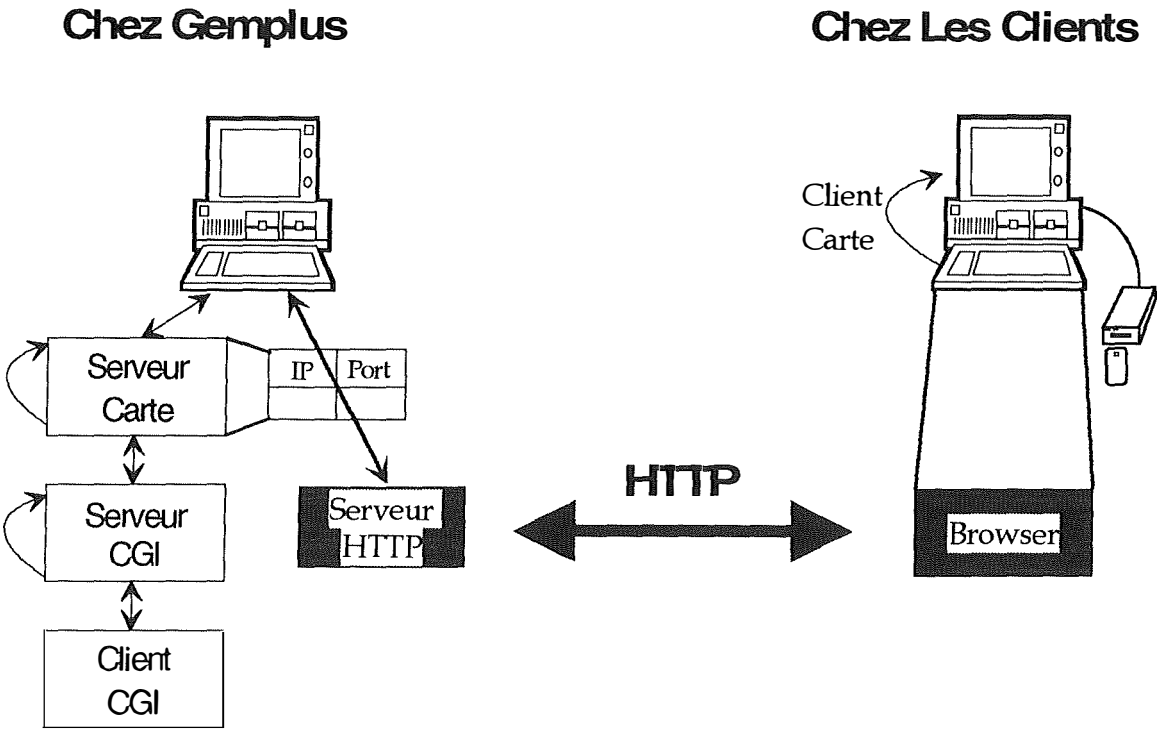


Figure 54 : L'architecture TLI

Le plus grand avantage de l'architecture TLI est de diminuer la charge pour le client. Contrairement à la première architecture, il ne doit plus posséder Orbix. Seul le processus client/carte est nécessaire. Ce processus est déclenché dès l'instant où le client désire se connecter à la carte. A partir de ce moment, il tourne et attend les requêtes. Il ne cessera son activité que s'il reçoit une requête de déconnexion.

Par contre, c'est chez le Serveur que se trouveront le serveur/carte, le serveur/CGI et le client/CGI. Le serveur/carte et le serveur/CGI sont deux processus qui tournent en permanence en attendant les clients potentiels. Le serveur/carte gère une table qui effectue la correspondance entre le numéro du client (adresse IP) et le numéro de port attribué à la communication. A chaque client/carte correspond un numéro de port.

L'arrivée d'un client/carte constitue la deuxième étape. C'est à cet instant que la table des correspondances est mise à jour. Le processus client/carte tournera jusqu'à la fin de l'utilisation de la carte par le membre du personnel hospitalier.

La dernière étape constitue l'exécution des commandes CQL. Lorsque, par exemple, le médecin consulte la carte à travers le browser. Les requêtes CQL sont envoyées par l'intermédiaire du CGI au processus client/CGI. Lui-même les transmet au processus serveur/CGI qui les envoie au processus serveur/carte. Celui-ci consulte sa table des correspondances pour voir si le client/carte est effectivement connecté. Dans ce cas, l'information est transmise au processus client/carte qui pourra accéder aux informations situées sur la carte. Il transfert celles-ci dans le sens inverse. Le client/CGI les reçoit, les transmet au CGI et se clôture.

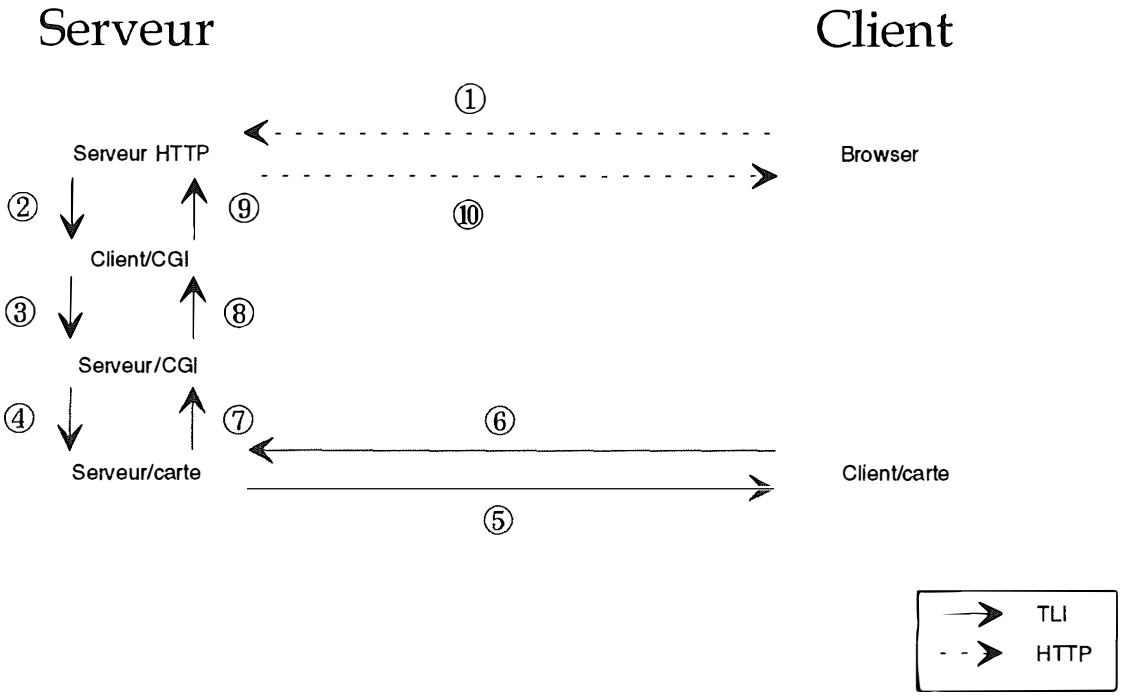


Figure 55 : Le schéma de communication via les TLI

La *Figure 56* illustre d'une façon plus précise la liaison entre les clients et le serveur. Lorsque Gemplus lance son serveur/carte la première fois, ce serveur est un utilisateur TLI à qui le fournisseur de transport attribue un point de transport (tep) unique. Un client, qui désire se connecter au serveur, utilisera ce point de transport pour lui envoyer sa demande de connexion. Si le serveur accepte la connexion, il attribue au client un numéro de port. Si un autre client effectue cette même demande, il se verra attribuer un numéro de port différent du précédent.

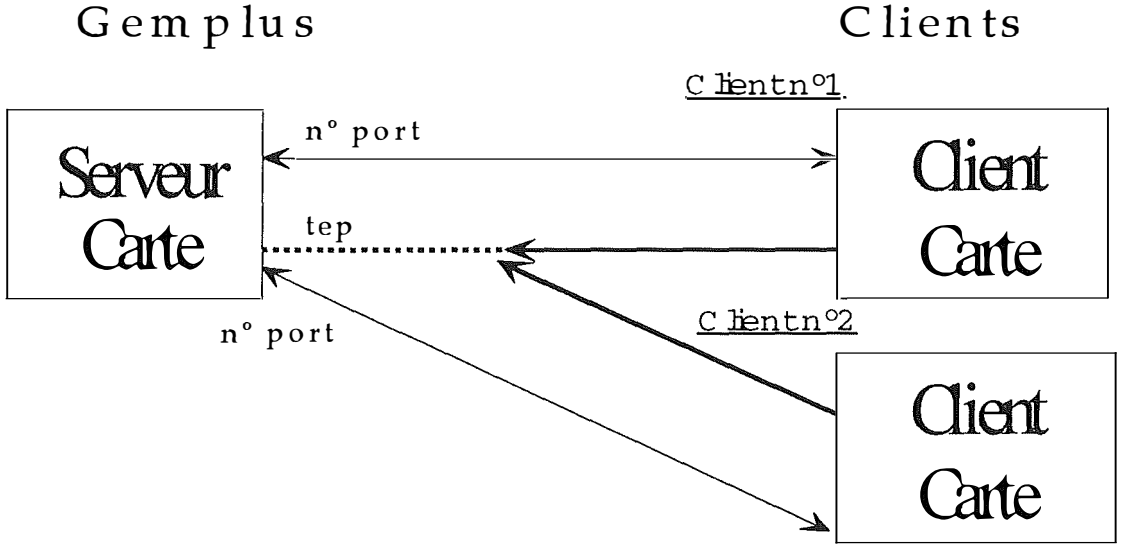


Figure 56 : La liaison avec les clients

4. Conclusions

La sécurité dans le monde de l'Internet est et restera une grande question dans les années à venir. Nous avons toutefois décidé de vous présenter un protocole existant actuellement sur le marché. Nous espérons avoir apporté un certain éclairage dans ce domaine. L'intégration de la carte à microprocesseur avec de tels protocoles pourrait peut-être se révéler intéressante. Ne pourrait-on pas utiliser la carte, qui bénéficie d'une confiance importante du point de vue de la sécurité, afin de stocker les clés les plus sensibles ?

Dans la seconde section, nous avons montré qu'il était possible de remplacer la plate-forme Orbix par des sockets Unix. Suite à notre présentation, nous pouvons nous rendre compte que **les mécanismes proposés par les TLI sont relativement simples et efficaces**. Ils représentent néanmoins le désavantage d'être un standard propriétaire. Notre objectif n'était toutefois que de montrer la faisabilité d'un tel projet. L'étape suivante serait de développer le même genre d'architecture pour les systèmes d'exploitation les plus utilisés chez les particuliers : Dos, Windows 95 et Windows NT.

Conclusion générale

D'un point de vue personnel, ce travail s'est avéré être très enrichissant. En effet, il nous a donné l'occasion d'une part de mettre en pratique des techniques et des méthodes que nous avions apprises à l'Institut (l'approche transformationnelle par prototypage, la découpe en unités de présentations, ...), et d'autre part d'aborder un grand nombre de domaines qui nous étaient moins connus auparavant.

Dans ce travail, nous avons voulu partager avec le lecteur la satisfaction que procure la découverte de différents mondes hétérogènes et le plaisir de se rendre compte que ces mondes peuvent coopérer au sein d'une même application.

En effet, pour réaliser notre application WebSanté, nous avons commencé à étudier globalement les cartes à microprocesseur. Ensuite, nous avons approfondi une carte en particulier : la carte CQL. L'objectif étant de manipuler la carte WebSanté avec un browser de l'Internet, nous nous sommes intéressés de très près au fonctionnement du Web. Les CGI nous ont permis d'intégrer le monde de la carte et de l'Internet.

Dans ce cadrer, nous nous sommes rendus compte de deux problèmes essentiels : la sécurité sur l'Internet et la plate-forme Orbix. Nous avons donc proposé une solution pour chacun de ces problèmes.

Après avoir pris un petit peu de recul par rapport à notre application, nous pensons que l'on pourrait y apporter certaines améliorations. Tout d'abord, la place disponible en mémoire reste limitée. Nous avons effectué certains tests et il apparaît que la carte peut contenir une moyenne de 15 lignes par table; c'est à la fois peu et beaucoup. Dans un certain sens, on peut dire que le patient ne fréquentera sûrement jamais 15 hôpitaux différents. Par contre, il pourrait subir tout au long de sa vie beaucoup d'actes élémentaires. Si l'on envisage de donner une carte à un patient pour toute son existence, la place mémoire pourrait devenir un facteur critique.

C'est pourquoi, dans cette hypothèse, nous proposerions de supprimer des informations moins pertinentes telles que les références des hôpitaux et des services.

Ensuite, un effort devrait avoir lieu en ce qui concerne l'ajout de nouveaux documents par le personnel médical. La consultation des données contenues dans la carte est très conviviale; la plupart du temps, un simple clic suffit. Par contre, il serait intéressant de fournir des outils au personnel médical pour créer leurs documents HTML. Des aides contextuelles pourraient également faciliter la tâche de l'utilisateur.

La sécurité des communications sur l'Internet est bien évidemment un point crucial à améliorer avant d'envisager une quelconque utilisation de ce produit. Les patients n'accepteraient jamais de voir circuler librement sur l'Internet leurs données médicales. De plus, d'un point de vue légal, les données à caractère médical sont des donnés sensibles et bénéficient d'une grande protection. Il s'agit de tenir compte de ce contexte.

Dans la deuxième partie du dernier chapitre, nous avons développé un modèle Client/Serveur basé sur les sockets Unix afin de supprimer la dépendance de notre application avec Orbix. Il faudrait maintenant appliquer le schéma que nous avons proposé aux autres environnements courants : DOS, Windows 95 et Windows NT.

Nous espérons vous avoir transmis notre conviction que les dossiers portables sur carte à microprocesseur ont un grand avenir devant eux. La technologie des cartes à microprocesseur ne cesse de s'améliorer. Par exemple, des zones mémoires de 32Koctets seront bientôt envisageables et elles ouvriront la porte à des cartes multi-applicatives. Carte de débit, carte de crédit, porte-monnaie électronique et dossiers portables pourraient se trouver sur une même carte à microprocesseur ...

Table des abréviations

AM	Application Manager
CERIM	Centre de Recherche en Informatique Médicale
CERN	Centre Européen de Recherche Nucléaire
CGI	Common Gateway Interface
CI	Card Issuer
CNRS	Centre National de Recherche Scientifique
CORBA	Common Object Request Broker Architecture
CQL	Card Query Language
DES	Data Encryption Standard
DN	Distinguished Name
EEPPROM	Electricaly Erasable and Programmable Read Only Memory
EPROM	Ecletrically Programmable Read Only Memory
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDL	Interface Definition Language
IP	Internet Protocol
ISO	International Organization for Standardization
MAM	Microcalculateur Autoprogrammable Monolithique
MD5	Message Digest 5
NICC	National Information Control Centre
OMG	Object Management Group
RAM	Random Access Memory
RD2P	Recherche et Développement Dossier Portable
ROM	Read Only Memory
RSA	Rivest, Shamir et Adelman
SGBD	Système de Gestion de Base de Données
S-HTTP	Secure HyperText Transfer Protocol
SPOM	Self-Programmable One-Chip Microcomputer
SQL	Standard Query Language
SU	Standard User
TCP	Transmission Control Protocol
TEP	Transport EndPoint
TLI	Transport Layer Interface
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WWW	World Wide Web

Les références bibliographiques

[BHLPVZ94] F. Bodart, A.-M. Hennebert, J.-M. Leheureux, I. Provot, J. Vanderdonckt, and G. Zucchinetti. *Dimensions clé pour une méthodologie de développement d'applications interactives.* International Eurographics Workshop on Design, Specification and Verification of Interactive Systems, Bocca di Magra (La Spezia), 8-10 june 1994.

[BLC95] Tim Berners-Lee and Daniel W. Connolly. *Hypertext Markup Language - 2.0.* Internet Network Working Group RFC 1866, MIT/W3C, November 1995. <URL:ftp::://ds.internic.net/rfc/rfc1866.txt>.

[BLFF95] Tim Berners-Lee, Roy T. Fielding, and Henry Frystyk Nielsen. *HyperText Transfer Protocol - HTTP/ 1.0 IETF draft* - Work in progress (expires April 14, 1996), MIT/LCS, UC Irvine, CERN, October 14, 1995. <URL:http://www.w3.org/hypertext/WWW/Protocols/HTTP1.0/draft-ietf-http-v10-spec-04.txt>

[BLMM94] Tim Berners-Lee, Larry Masinter, and Mark McCahill. *Uniform Resource Locators (URL).* Internet Network Working Group RFC 1738, CERN, Xerox PARC, University of Minnesota, December 1994. <URL:ftp::://ds.internic.net/rfc/rfc1738.txt>.

[Gem93] Gemplus Card International, BP 100, 13881 Gémenos cedex, France. *CQL Card Reference Manual*, preliminary version edition, April 1993.

[Grim92] Georges Grimonprez. *Etude et réalisation d'une carte à microprocesseur intégrée aux SGBDs.* Memoire d'habilitation à diriger des recherches. Université des Sciences et Technologies de Lille, Cité Scientifique, 59655 Villeneuve D'Ascq cedex, France, 1992.

[GUQ91] Louis C. Guillou, Michel Ugon, and Jean-Jacques Quisquater. *The Smart Card, a standardized security device dedicated public cryptology.* 1991.

[ISO94] ISO International Organization for Standardization, P.O. Box 56, 1211 Geneva 20, Switzerland. *International Standard ISO/IEC 9075 : Inforamtion Technology - Database - languages - SQL*, third edition, Novembre 1992.

[JJV95] Jean-Jacques Vandewalle. *Cours cartes à microprocesseur.* IUT "A" de Lille I, Département Informatique, Cité Scientifique, 59655 Villeneuve D'Ascq cedex, France. 1995.

[LeWeb95] Kevin Hughes. Entering the World Wide Web : a Guide to Cyberspace. Entreprise Integration Technologies, May 1994.

[McC95] R. McCool. *The Common Gateway Interface.* NCSA, 1995. <URL:http://hoohoo.ncsa.uiuc.edu/cgi/overview.html>.

[MGG96] Philippe Merle, Christophe Gransart, and Jean-Marc Geib. CorbaScript and CorbaWeb : A Generic Object-Oriented Dynamic Environment upon CORBA. In proceedings of TOOLS Europe'96, Palais des Congrès, Paris, France, February 26-29, 1996. Prentice-Hall, February 1996.

[NgBoSa94] NGUYEN N.T., BOSLY A. & SADAOUI S. *Hemacard, a pilot poject in Belgium*, in Proceedings of the 5th International Congress on Health Cards, Venice, May 1994.

[OHE96] Robert Orfali, Dan Hakey, and Jeri Edwards. *The Essential Distributed Objects Survival Guide*, p. 1-107. Wiley, 1996.

[Para86] Pierre Paradinas. Mémoire d'habilitation à diriger des recherches. Université des Sciences et Technologies de Lille, Cité Scientifique, 59655 Villeneuve D'Ascq cedex, France, 1986.

[ReSc4] E. Rescorla, and A. Schiffman.*The Secure Hypertext Transfer Protocol.* Enterprise Internet Draft (expires 5/95), Integration Technologies, December 1994.
<URL:http://www.eit.com/projects/s-http/shttp.txt >.

[Sab93] Sébastien Sabre. *Etude de la mise en place d'une OpenCard en milieu hospitalier. Rapport de stage effectué au CERIM*, Ecole d'ingénieur EUDIL, Cité Scientifique, 59655 Villeneuve D'Ascq cedex, France, June 1993.

[SECU95] Joël Hubin. La sécurité en informatique, notes utilisées dans le cadre du cours "Sécurité et fiabilité des systèmes informatiques". Draft, february 1995.

[SHT95a] *Secure Hyertext Transfer Protocol*, version 0.3. 1995.
<URL:http://www.commerce.net/software/SHTTP/Docs/
SHTTP.référence.html>.

[SMos95] *Secure NCSA Mosaic*. 1995.
<URL:http://www.commerce.net/software/SMosaic/Docs/SMosais.référence.
html>.

[SunOS40] Sun Microsystems. Chapter 9 : Transport Level Interface Programming, p. 189-249. Revision A, of 27 March 1990.

[Durif94] Philippe Durif. Outils de Communication Réseau sous Unix BSD 4.x. Université des Sciences et Technologies de Lille, Cité Scientifique, 59655 Villeneuve D'Ascq cedex, France, September 1994.

Annexe A

Les principales commandes CQL

Le tableau ci-dessous présente les trois catégories de commandes. Dans les paragraphes suivants, nous détaillerons les commandes les plus utilisées.

Gestion des utilisateurs	Définition des structures et des accès aux données	Manipulation des données
create application	create table	declare cursor
create user	create view	open
present	create dictionary	fetch
change password	drop table	fetch_next
unlock	drop view	insert
delete user	grant	erase
create key	revoke	updatec
status		begin transaction
authenticate		commit
check		rollback
		read record

Tableau 8 : Les commandes CQL

a) Les commandes de gestion des utilisateurs

CREATE APPLICATION

CREATE APPLICATION <nom du gestionnaire> <taux de ratification> <mot de passe>;

< nom du gestionnaire> : un identificateur
<taux de ratification> : un nombre

Cette commande définit un profil de gestionnaire d'application. Le nom et le mot de passe du gestionnaire d'application devront être introduits lors de l'ouverture d'une session. Le taux de ratification représente le nombre de fois que le gestionnaire peut se présenter avec un mot de passe erroné avant que la carte ne le lui interdise. Si le taux de ratification est nul, cela signifie qu'il n'y a pas de limitation.

CREATE USER

CREATE USER <nom d'utilisateur> <taux de ratification> <mot de passe>;

<nom d'utilisateur> : un identificateur

Cette commande permet de déclarer un utilisateur et est réservée à l'émetteur et aux gestionnaires d'application.

PRESENT

PRESENT <nom d'utilisateur> <mot de passe>;
PRESENT PUBLIC;

La commande PRESENT peut être utilisée par n'importe quel utilisateur et a pour but d'ouvrir une session. Pour que celle-ci soit ouverte, il faut que le mot de passe soit valide et que l'utilisateur ne soit pas bloqué (suite à un dépassement de son taux de ratification). Une ouverture de session signifie qu'un contexte est créé en fonction du type de l'utilisateur.

Chaque présentation erronée augmente le compteur de ratification de une unité. Quand celui-ci atteint le taux de ratification prévu à la création de l'utilisateur, l'utilisateur est bloqué. Le créateur de cet utilisateur est le seul habilité à débloquer l'utilisateur. Par contre, le compteur de ratification est remis à zéro quand l'utilisateur se présente avec succès.

La commande PRESENT PUBLIC ne nécessite aucun mot de passe, étant donné que son accès à la carte sera restreint.

CHANGE PASSWORD

CHANGE PASSWORD <ancien mot de passe> <nouveau mot de passe>;

UNLOCK

UNLOCK <nom d'utilisateur>;

DELETE USER

DELETE USER <nom d'utilisateur>;

CREATE KEY, AUTHENTICATE ET CHECK

Ces trois commandes sont utilisées si l'algorithme du DES est présent dans la carte. Elles permettent une authentification à clé privée. Nous ne les détaillerons pas étant donné que les cartes dont nous disposions pour notre application ne disposaient pas du DES.

STATUS :

STATUS;

Cette commande renvoie le numéro de série de la carte et la mémoire EEPROM utilisée dans la carte.

b) Les commandes de structures et d'accès aux données

CREATE TABLE

CREATE TABLE <nom de la table> (<liste des colonnes>);

<nom de la table> : identificateur
<liste des colonnes> : < identificateur >[,< identificateur >...]
Exemple :
create table ETAT_C(NOM,PRE,NAIS,NAT);

CREATE VIEW

CREATE VIEW <nom de la vue> AS <définition de la vue>;

<nom de la vue> : identificateur
<définition de la vue> : <une clause select>

Exemple :
create view NAT_B as select NOM,PRE from ETAT_C where NAT = 'B';

CREATE DICTIONARY

CREATE DICTIONARY <nom générique des dictionnaires>;

<nom générique des dictionnaires> : un identificateur de maximum quatre caractères

Cette commande crée trois vues sur les tables systèmes. Les noms de ces vues seront le nom générique auquel on a ajouté respectivement _T, _U, _P pour les vues concernant les tables, les utilisateurs et les privilèges.

Exemple :
create dictionary DICO;
select * from DICO_T;

Drop table et drop view :

DROP TABLE <nom de la table>;
DROP VIEW <nom de la vue>;

GRANT ET REVOKE

GRANT <liste de privilèges> ON <nom de table, vue ou dictionnaire> TO <nom d'utilisateur>;
REVOKE <liste de privilèges> ON <nom de table, vue ou dictionnaire> TO <nom d'utilisateur>;

<privilège> : insert | update | delete | select | all
<liste de privilèges> : <privilège>[,<privilège>]

La commande GRANT permet d'attribuer des privilèges à un utilisateur sur un objet. Par contre, la commande REVOKE retire des privilèges à un utilisateur.

c) Les commandes de manipulation de données

SELECT

SELECT * | <liste de colonnes> FROM <nom de l'objet> [where <condition de recherche>];

<liste de colonnes> : identificateur [,identificateur ...]
<nom de l'objet> : identificateur d'une table, d'une vue ou d'un dictionnaire

La commande SELECT est utilisée pour accéder à un sous-ensemble d'une table, d'une vue ou d'un dictionnaire. Les colonnes sélectionnées doivent appartenir à la table en question et il est autorisé de sélectionner plusieurs fois une même colonne. De plus l'ordre dans lequel on place les colonnes sélectionnées ne doit pas nécessairement être le même que l'ordre défini dans l'objet.

SELECT est la commande de sélection la plus naturelle dans le langage SQL. Elle n'est toutefois pas implémentée dans la carte CQL. Ce qui ne signifie pas que l'on ne puisse pas l'utiliser. En effet, cette instruction sera décomposée avant d'accéder à la carte en instructions plus élémentaires (DECLARE CURSOR, FETCH, NEXT, ...).

DECLARE CURSOR

DECLARE CURSOR AS <sélection>;

<sélection> : clause select

Un curseur est déclaré pour définir une sélection de lignes et de colonnes d'une table, d'une vue ou d'un dictionnaire. Un seul curseur peut être déclaré à la fois. Cette commande est purement syntaxique, c'est-à-dire qu'elle ne change pas le contenu des données. C'est uniquement un pointeur sur les données satisfaisant la clause select.

OPEN

OPEN;

Cette commande positionne le curseur sur la première occurrence des données qui satisfont à la sélection du DECLARE CURSOR.

NEXT

NEXT;

NEXT positionne le curseur sur l'élément suivant qui satisfait à la sélection du DECLARE CURSOR.

FETCH, FETCH_NTEXT ET READ RECORD

Exemple :

```
declare cursor as select NAT from ETAT_C where NOM='pierre';
open;
fetch_next;
read record;
fetch_next;
read record;
```

Fetch et fetch_next sont utilisés pour préparer la cellule à être envoyée au système. Read record permet d'obtenir la cellule concernée.

INSERT

INSERT INTO <nom de table> VALUES (<liste de chaînes de caractères>);

<liste de chaînes de caractères> : chaîne de caractères [,chaîne de caractères ...]

Cette commande est utilisée pour insérer une ligne dans une table. L'utilisateur qui exécute cette commande doit posséder le privilège. Comme il s'agit d'insérer une ligne entière d'une table, toutes les colonnes doivent être présentes dans la liste des chaînes de caractères. La taille du tampon de la carte (64 octets) limite l'insertion des données à cette taille. Il convient alors de faire une insertion avec certaines colonnes vides (") et d'effectuer une mise à jour par la suite; comme l'illustre l'exemple suivant.

Exemple :

```
insert into TABLE1 values ('undeuxtrois','quatrecinq','sixsepthuit',");
updatec set COL4='neufdixonze';
```

ERASE :

ERASE;

La commande ERASE supprime une ligne d'une table. Pour cela, un curseur doit être déclaré sur cette ligne auparavant (DECLARE CURSOR). L'utilisateur qui désire supprimer une ligne doit posséder le privilège DELETE.

UPDATEC

UPDATEC SET <nom de colonne> = 'chaîne de caractères' [,<nom de colonne> = 'chaîne de caractères'];

Cette commande sert à mettre à jour les données d'une ou plusieurs colonnes d'une même table. Avant d'exécuter celle-ci, il faut avoir déclaré un curseur. L'utilisateur de cette commande doit posséder le privilège UPDATE.

BEGIN TRANSACTION, COMMIT ET ROLLBACK

Ces trois commandes permettent la mise en place du mécanisme de transaction. Une transaction doit satisfaire quatre critères : l'atomicité, la cohérence, l'indépendance et la durabilité. L'atomicité signifie que l'exécution sera entièrement réalisée ou pas du tout. La cohérence garantit que la base de données restera, après l'exécution de la transaction, dans le même état de cohérence qu'avant son exécution. L'indépendance suppose que la transaction est exécutée sans tenir compte de son environnement. Enfin, la durabilité assure que si la transaction parvient à son terme, les mises à jour sont permanentes.

Nous ne détaillerons pas davantage ces trois commandes car nous ne pouvons les utiliser dans notre application. En effet, elles ne sont disponibles que si la personnalisation de la carte l'a prévu.